Instituto de Matemática e Estatística - USP

Trabalho de Formatura Supervisionado

Crea+ Computação



Aluno:
William A. Miura Gnann

Supervisor:
José Coelho de Pina

2 de dezembro de 2013

Sumário

1 Informações Gerais **5**
- 1.1 Crea+ . . . 5
- 1.2 Sem computadores . . . 5
- 1.3 Com computadores . . . 6
  - 1.3.1 Turtle . . . 6
  - 1.3.2 Pygame . . . 7
  - 1.3.3 Recursos . . . 7

2 Aula 1: Gênese **8**
- 2.1 Atividade: O Computador que Anda para Frente . . . 8
  - 2.1.1 Objetivos . . . 8
  - 2.1.2 Descrição . . . 8
- 2.2 A História Antiga da Computação . . . 9
  - 2.2.1 Objetivos . . . 9
  - 2.2.2 Calculadoras Mecânicas . . . 9
  - 2.2.3 Teares Programáveis . . . 10
  - 2.2.4 Máquina Analítica . . . 10
  - 2.2.5 Pós-eletrônica: o ENIAC . . . 10
- 2.3 Modelo Teórico . . . 11
- 2.4 Relatório . . . 14

3 Aula 2: Binários e *Bits* **15**
- 3.1 Atividade: O Campeonato de Binário . . . 15
  - 3.1.1 Objetivos . . . 15
  - 3.1.2 Descrição . . . 15
- 3.2 Os Binários na Computação . . . 16
- 3.3 Relatório . . . 18

4 Aula 3: Lógica e Portas **19**
- 4.1 Atividade: Detetive . . . 19
  - 4.1.1 Objetivos . . . 19
  - 4.1.2 Descrição . . . 19
- 4.2 Um pouco de Lógica . . . 19
- 4.3 Portas Lógicas . . . 21
- 4.4 Relatório . . . 23
- 4.5 Caso 1: a melhor banda de todos os tempos da última semana . . . 24
  - 4.5.1 Histórico . . . 24
  - 4.5.2 Enunciado . . . 24
  - 4.5.3 Informações . . . 24
- 4.6 Caso 2: reforços para o Brasileirão . . . 25
  - 4.6.1 Histórico . . . 25

4.6.2 Informações . . . 25
4.7 Desafio: O Teste de "Einstein" . . . 26
4.7.1 Histórico . . . 26
4.7.2 Enunciado . . . 26
4.7.3 Informações . . . 27

5 Aula 4: Códigos e Criptografia 28
5.1 Atividade: Criptoanálise . . . 28
5.1.1 Objetivos . . . 28
5.1.2 Descrição . . . 28
Anevm, anevm, r anevm . . . 28
5.2 Cifra de César . . . 29
5.2.1 Gaius Julius Caesar . . . 29
5.2.2 A cifra . . . 29
5.2.3 Como quebrar a cifra? . . . 30
5.3 Códigos . . . 30
5.3.1 ASCII . . . 30
5.4 Relatório . . . 32

6 Aula 5: Introdução ao *Python* 33
6.1 Python . . . 33
6.2 Interpretador . . . 33
6.3 Primeiro Contato . . . 34
6.3.1 *print* . . . 34
6.3.2 Operadores . . . 34
6.4 Problemas . . . 36
6.5 Relatório . . . 37

7 Aula 6: Variáveis e Tipos 38
7.1 Problema da lojinha - versão 1 . . . 38
7.2 Variáveis . . . 39
7.3 Problema da lojinha - versão 2 . . . 40
7.4 Tipos de dados . . . 40
7.5 Relatório . . . 42

8 Aula 7: Entrada de dados 43
8.1 *input* . . . 43
8.2 Problema da lojinha - versão 3 . . . 44
8.3 Relatório . . . 45

9 Aula 8: Implementação da Cifra de César 46
9.1 Revisão . . . 46
9.1.1 Cifra de César . . . 46
9.1.2 ASCII . . . 47
9.2 Versão 1 . . . 47
9.3 Pizza . . . 48
9.4 Versão 2 . . . 49
9.5 Versão 3 . . . 50
9.6 Relatório . . . 51

10 Aula 9: Corrida de Tartaruga! 52
10.1 Atividade: Corrida de Tartaruga! . . . 52
10.1.1 Objetivos . . . 52
10.1.2 Descrição . . . 52
10.1.3 Regras . . . 52
10.1.4 Instruções . . . 52
10.2 Relatório . . . 54
10.2.1 Editor de texto . . . 54
10.2.2 $14 * 2 = 28$ . . . 54
10.2.3 Tk . . . 54

11 Aula 10: Exercícios 55
11.1 *print* . . . 55
11.2 Operadores . . . 55
11.3 Variáveis . . . 55
11.4 Relatório . . . 58

12 Aula 11: Condicionais 59
12.1 *if* . . . 59
12.2 Condições . . . 60
12.3 Lógica (de novo!) . . . 61
12.4 *if*, *else* e *elif* . . . 62
12.5 Relatório . . . 63

13 Aula 12: Jogo da Velha 64
13.1 Informações gerais . . . 64
13.2 Modelo . . . 64
13.3 Jogo . . . 65
13.3.1 Turno . . . 65
13.3.2 Vitória . . . 65
13.3.3 Velha . . . 65
13.3.4 Troca . . . 65
13.4 Relatório . . . 66

14 Conclusão 68

15 Parte subjetiva 69
15.1 Desafios e Frustrações . . . 69
15.2 Disciplinas . . . 70
15.2.1 Relevantes para o TCC . . . 70
15.2.2 Relevantes . . . 70
15.2.3 Irrelevantes . . . 71
15.3 Continuação . . . 72

Introdução

Com o advento da era da informática, muitas das atividades, que outrora eram feitas pessoalmente, passaram a ter uma opção virtual. O computador hoje é uma realidade. O mundo está conectado!

A Computação passou a ser algo como português ou aritmética. Contudo, o ensino básico ainda não acompanhou essa transição veloz do papel para os *bits*. O senso comum diz que Computação é operar computadores.

Num contexto onde cada vez mais o computador e a Computação fazem parte da vida das pessoas parece importante ter contato com esse tipo de conteúdo nas escolas.

O objetivo deste trabalho é justamente montar e aplicar um curso de computação voltado ao público do ensino básico.

Capítulo 1

Informações Gerais

A atividade de Computação começou a ser oferecida na Escola Estadual Odon Cavalcanti em agosto de 2013 a partir do projeto Crea+.

O curso teve duração de quatro meses sendo dividido em treze aulas, parte em laboratório, parte em sala de aula. O objetivo do curso é **dar uma noção intuitiva** de como "pensa" um computador.

Nesta versão do curso, a quantidade de alunos presentes por aula esteve em torno de 10.

1.1 Crea+

O Crea+[3] é um projeto de voluntariado com raízes no Chile cujo objetivo é contribuir com a educação pública complementando a formação dos alunos. O programa de aula consiste em reforço de matemática e atividades socioculturais.

As aulas são dadas aos sábados das 10h00 às 13h00 dividindo-se em 80min de reforço de matemática, 20min de lanche e 80min de atividades socioculturais.

Atualmente o projeto trabalha com alunos do Ensino Fundalmental Ciclo II em duas escolas de São Paulo:

- EE Prof. Daniel Paulo Verano Pontes;
- EE Prof. Odon Cavalcanti.

Dentre as diversas atividades socioculturais que o Crea+ oferece aos alunos participantes, encontramos, além da que norteia este trabalho, artes, culinária, dança, música, esportes etc.

1.2 Sem computadores

Inicialmente a proposta do curso era ser inteiramente sem computadores baseando-se, em princípio, no Computer Science Unplugged.

Contudo, houve uma modificação na proposta e no planejamento do curso para termos aulas com computadores.

O cerne da seção do curso sem computadores é mostrar de uma maneira lúdica como o computador funciona desde os bits até os programas.

Começamos entendendo de onde e por que surgiu um computador. Parece interessante dar uma ideia aos alunos qual a conjuntura do mundo quando presenciamos a gênese dessas máquinas. Em seguida, visitamos os números binários e vimos sua relação com o *bit*. Também vimos como funciona um *bit* dentro de um computador.

Foi visto, também, o conceito de portas lógicas para dar uma ideia de como podemos transformar esses *bits*. Parece bastante razoável os alunos terem uma noção do que é o processador. Depois das portas, falta entender, de maneira geral, como o aglomerado de zeros e uns consegue ser visto como uma imagem ou este próprio texto.

As aulas dessa parte do curso foram divididas geralmente em três momentos:

1. Atividade envolvendo alguns conceitos;
2. Exposição dos conceitos;
3. Exercícios.

A parte sem computador do curso foi ministrada em sala de aula convencional com a ajuda de voluntários do Crea+, sobretudo nas primeiras. No curso deste semestre, tivemos cinco dessas aulas.

1.3 Com computadores

Diferente da abordagem "por quê" do módulo sem computador do curso, o módulo com o computador trata de "como" escrever programas. Para tanto, utilizamos Python.

Python tem diversos aspectos interessantes como uma linguagem introdutória, dentre eles:

- interatividade;
- simplicidade;
- portabilidade;
- Turtle;
- Pygame.

O principal objetivo é dar uma noção aos alunos de como dar ordens ao computador. Para tentar atingir esse objetivo, vimos, por exemplo, como imprimir na tela, operadores, variáveis, tipos, condicionais etc.

O modelo das aulas foi baseado em experimentação e exercícios e o fato de Python ser uma linguagem interpretada ajudou bastante.

1.3.1 Turtle

Turtle é uma biblioteca do Python, nativa na maioria das versões, escrita a partir do TkInter - a interface gráfica padrão do Python.

É uma forma lúdica de introduzir programação ao público infantil baseando-se no objetivo de movimentar uma tartaruga digital pela tela.

A partir da movimentação da tartaruga, podemos explorar diversas atividades como desenhar formas geométricas, apostar corrida etc. Inclusive o Turtle foi utilizado para fazer uma corrida de tartarugas.

1.3.2 Pygame

Pygame é um conjunto de módulos que trabalha sobre o SDL - uma biblioteca que provê acesso a diversos recursos de multimídia - e, como o próprio nome já diz, também serve para desenvolver jogos.

Os recursos multimídia são bastante desejáveis, pois dão uma motivação a mais por permitirem a criação de programas com razoável interação com o usuário.

No escopo deste projeto, utilizamos o Pygame para confeccionar uma versão digital para o joguinho de cartas binárias e uma versão interagível do Jogo da Velha.

1.3.3 Recursos

As aulas foram realizadas no laboratório de computadores utilizando uma distribuição Live do Debian GNU/Linux.

O laboratório conta com quatorze computadores comuns cuja configuração permite executarmos o Linux a partir do pen drive. Dada a quantidade média de alunos presentes, conseguimos deixar um aluno por computador.

A principal razão para se utilizar Linux a partir de um pen drive, em detrimento ao sistema operacional "nativo", é a flexibilidade de configuração e instalação de programas sem ter de passar pelo crivo do responsável pelo laboratório. Além disso, dá, mesmo que indiretamente, uma certa experiência com um sistema operacional novo.



Como ambiente de desenvolvimento, utilizamos o IDLE. Sua escolha se deu por se tratar de um ambiente mais amigável que o *shell* do Python e por ter sido construído a partir do TkInter que é padrão, por fim, alguns pacotes de Python contém o IDLE.

Capítulo 2

Aula 1: Gênese

O que é um computador? Sem sombra de dúvidas, muita gente se arrisca a responder a essa pergunta, mas quase nenhuma delas sabe de onde surgiu tal máquina.

Diversas invenções tiveram seu surgimento associado à tentativa de resolver algum problema vivenciado por seu inventor. Entretanto, muitas delas adquiriram usos cujo autor jamais imaginara.

O computador, de certa forma, é uma dessas invenções. Numa próxima seção veremos alguns detalhes interessantes acerca da invenção chamada computador.

Como funciona um computador? Outra pergunta cujo senso comum não consegue responder com facilidade. Na próxima seção teremos uma atividade que introduz alguns problemas de se trabalhar com um computador.

2.1 Atividade: O Computador que Anda para Frente

2.1.1 Objetivos

Os objetivos desta primeira atividade são:

- Estabelecer um contato com programação;
- Apresentar o conceito de *instrução*;
- Mostrar algumas limitações do computador.

2.1.2 Descrição

Subitamente, o professor virou um computador! Precisamos levá-lo até a porta da sala, mas ele só aceita determinados tipos de instruções.

Os alunos inicialmente não sabem que as únicas instruções que o computador aceita são:

- **E** (esquerda): o computador dará um passo com a perna esquerda para frente;
- **D** (direita): o computador dará um passo com a perna direita para frente.

E agora? O jeito é descobrirmos que instruções o *computador* aceita e depois tentarmos levá-lo para a porta.

Mas o que é uma instrução?

Instrução é uma espécie de ordem que o computador consegue interpretar. Nesse sentido, podemos tentar mandar, por exemplo, o computador andar até a porta. Já um *Programa* é um conjunto ordenado de instruções como, por exemplo, **EDEDEDED**.

Mesmo depois de os alunos descobrirem que o computador só compreende **E**s e **D**s, há outros problemas que o computador pode enfrentar. É razoável acreditar que os professores geralmente não conseguirão dar 64 passos com a perna esquerda antes de darem um passo com a perna direita.

Desta forma, vale, também, a seguinte regra:

o computador não pode dar mais que dois passos para frente com a mesma perna.

É interessante simular diversas vezes com a turma a caminhada do computador até a porta, inclusive recomenda-se chamar os alunos para ora registrarem as instruções, ora fazerem o papel de computador.

2.2 A História Antiga da Computação

Desde a Idade Antiga a humanidade se depara com problemas de aritmética. Diversos recursos foram desenvolvidos ao longo dos séculos para facilitar o fazer das contas. Ainda assim, fazer contas é uma tarefa com uma estrutura simples e repetitiva.

Nessa busca pela facilitação das contas, surgiram dispositivos como o ábaco e engenhocas como a *Pascaline*. Tudo isso para facilitar a vida de quem faz contas.

Até o começo do século XX, existiam pessoas cuja profissão era apenas fazer contas!

Mas qual a relação entre o computador e as contas? O termo *computador* veio da palavra *computare* que significa calcular.

2.2.1 Objetivos

- Conhecer a concepção do computador;
- Tentar entender com mais detalhes alguns conceitos;
- Compreender os problemas das épocas vistas;
- Introduzir a estrutura de um computador.

2.2.2 Calculadoras Mecânicas

Uma vez que o ancestral do computador é justamente uma calculadora, faz sentido conhecermos como eram feitas essas calculadoras antigamente.

É importante ressaltar que as calculadoras mecânicas nada mais eram do que automatizadores de fazer contas.

A ideia é bem simples: contar de um número até o outro; de uma forma bem parecida com o que fazemos quando estamos aprendendo a fazer conta. Ex: $3 + 4$: $3, 4, \ldots, 5, \ldots, 6, \ldots, 7$

Legal. Mas como fazer algum mecanismo que resolva esse tipo de conta? Uma possibilidade é um mecanismo de engrenagens parecido com o de um relógio.

Como fazer para descobrir que hora será daqui a 2h37min sem fazer conta? Basta rodar os ponteiros do relógio! Mas, no caso de uma máquina de fazer adição, o ponteiro dos minutos seria a "unidade" e o ponteiro das horas, a "dezena".

O movimento do ponteiro das horas subordinado à volta do ponteiro dos minutos se comporta de uma maneira bem parecida com o *vai-um* conhecido das adições. A diferença é que no relógio o que "vai" é uma hora e não uma dezena.

A partir de um mecanismo bastante parecido, Blaise Pascal, no século XVII, desenvolveu sua calculadora, a *Pascaline*. A máquina foi desenvolvida por Pascal para ajudar seu pai a fazer as contas da coleta de impostos.

Um exemplo do funcionamento da máquina pode ser visto aqui:

`http://www.youtube.com/watch?v=CJ7o-ir4R_E`.

2.2.3 Teares Programáveis

Em plena Revolução Industrial, Joseph Marie Jacquard inventou uma máquina de tecer cujos padrões de tecido eram definidos por *cartões perfurados*.

Os cartões perfurados controlavam as operações do tear! Controlavam as operações do tear! *Controlavam as operações!*

Apesar de ser uma máquina de fazer tecidos, o Tear de Jacquard tem uma importância fundamental no surgimento dos computadores, afinal, é um mecanismo que consegue produzir padrões de *saída*, os tecidos, dependendo de padrões de *entrada*, os cartões.

2.2.4 Máquina Analítica

A máquina de Charles Babbage foi praticamente o primeiro computador a ser criado. Unia os processos *automáticos* das calculadoras à possibilidade de *programar* uma sequência de comandos. Esses comandos eram padrões de furos num cartão perfurado.

A Máquina Analítica era completa a tal ponto que, a partir dos cartões, dispunha até de recursos condicionais[1]. O conteúdo dos cartões pode ser visto como o conjunto de *instruções* que a Máquina Analítica aceitava.

Ok. Temos uma máquina e as instruções. O que podemos fazer?

Uma vez que possuímos as instruções podemos escrever *programas* e simulá-los. Para isso, é necessária uma boa dose de concentração e um pouco de faz de conta para imaginar que existe um computador, que ele está recebendo as instruções e que ele está se comportando como o esperado.

Ada Lovelace, assistente de Babbage, é considerada a *primeira programadora*. Ela escreveu diversos programinhas para a Máquina Analítica.

Contudo, o que está escrito no primeiro parágrafo é "praticamente". O que faltou à invenção de Babbage para ela não ser considerada o primeiro computador?

Faltou simplesmente ficar pronta...

2.2.5 Pós-eletrônica: o ENIAC

Electronic Numerical Integrator And Computer, ou algo como Computador Eletrônico (podemos abandonar, por ora, a parte da integração numérica).

Essencialmente construído para cálculo bélico[2], o ENIAC era capaz de fazer 5000 contas de adição com números de 10 dígitos em *um segundo*![3]

Sua programação era feita ligando e desligando tubos de vácuo em lugares apropriados.

Ele era ENORME! Tinha mais de 27 toneladas e ocupava uma área de 167m^2 [4].

O que tem de tão importante o tal do ENIAC?

Diferentemente da Máquina Analítica, ele ficou pronto. Tão pronto que realmente foi utilizado para fazer contas, inclusive fazendo as contas de simulação da bomba de hidrogênio!

[1] Veremos isso com mais cuidado na primeira aula no laboratório.

[2] Referente a armamentos e coisas de guerra.

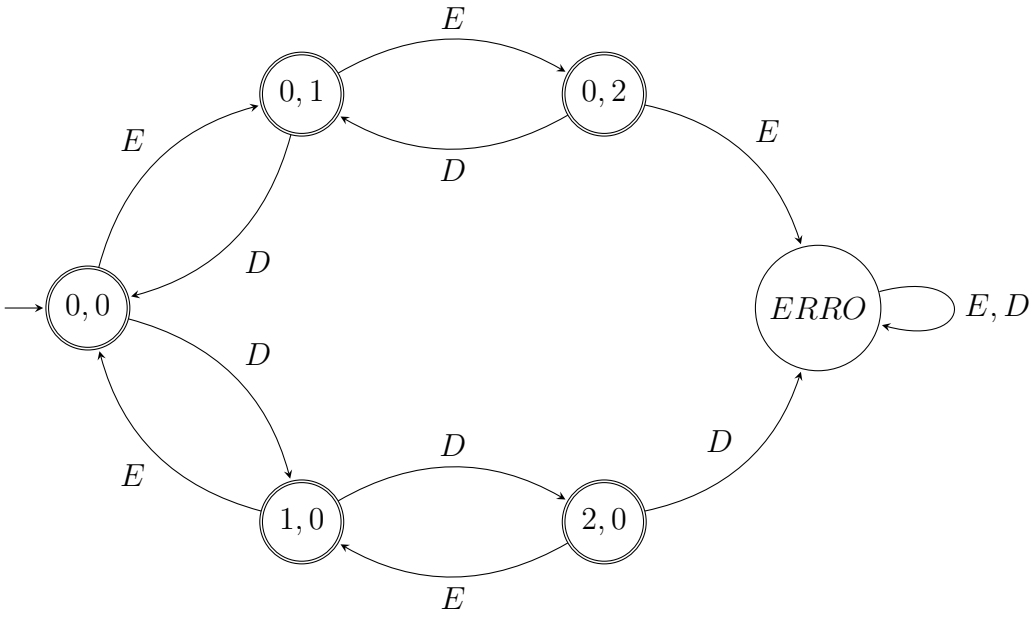
[3] Esse número é patético se comparado ao poder de processamento dos computadores modernos.

Principalmente, foi o primeiro computador completo. Teoricamente, o ENIAC consegue, de uma forma absurdamente lerda, fazer qualquer coisa que um computador moderno consiga fazer.

2.3 Modelo Teórico

Na década de 1930, houve um marco na História da Computação: pela primeira vez falava-se em Computação sem falar em computadores.

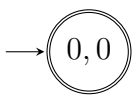
Muito disso se deve à criação do modelo teórico de Turing, a Máquina de Turing.



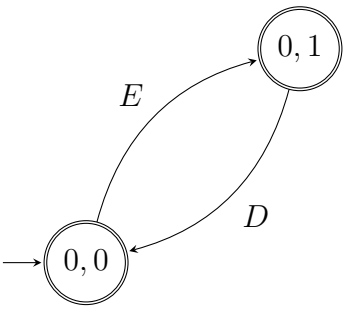
A Máquina de Turing, MT, é formada por:

- Um conjunto de *estados* (as bolinhas do desenho);
- Um conjunto de *transições* (as setinhas com as letras);
- Uma fita onde lemos uma posição por vez (a nossa sequência de **E** e **D** do começo da aula).

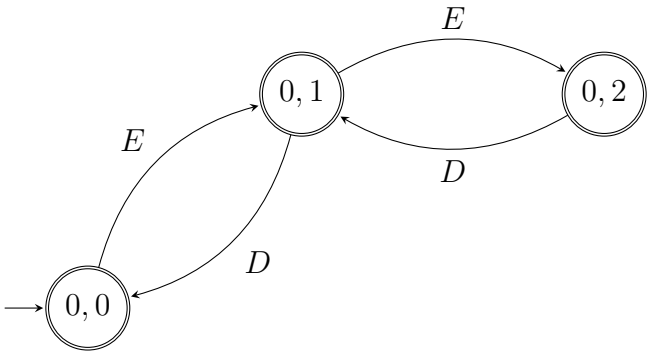
Vamos tentar construir a MT referente ao Computador que Anda para Frente.
Primeiramente, a fita já foi definida. São os **E**s e **D**s.
Agora precisaremos definir por onde começar.



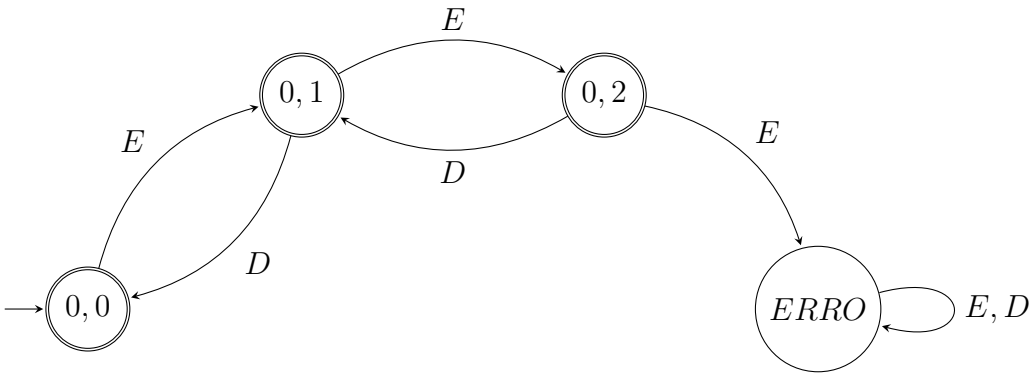
Vamos chamar de $(0, 0)$ porque é a diferença de passos entre as pernas esquerda e direita. Desta mesma forma, faz sentido colocarmos mais uma bolinha: a $(0, 1)$ que simboliza o passo para a esquerda.



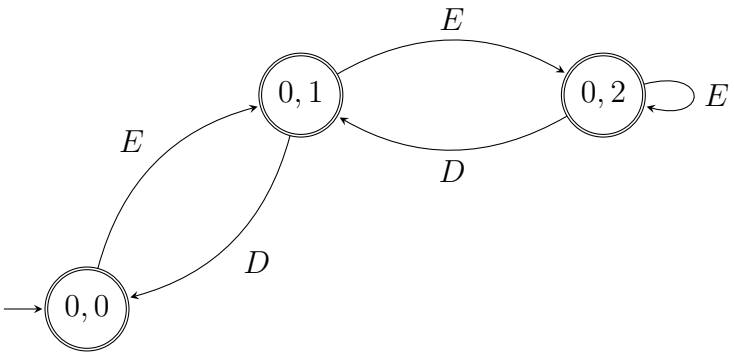
Como 2 é o máximo de número de passos, precisaremos da $(0, 2)$.



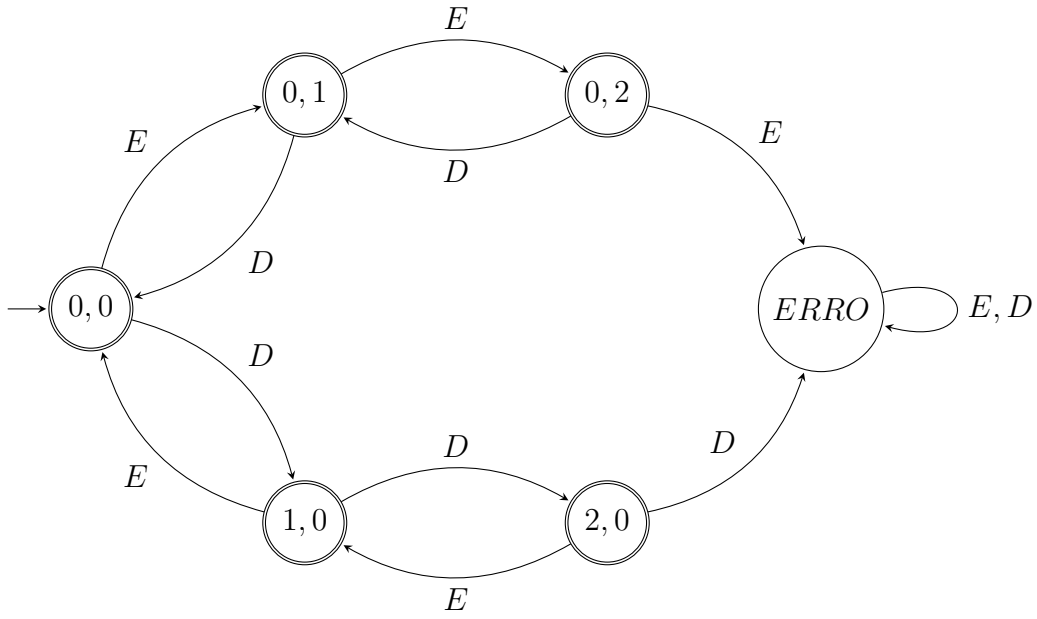
Se o "computador" tentar mais um passo para a esquerda, ele deverá quebrar. Podemos simbolizar isso acrescentando o $(ERRO)$.



Note que chegando no erro não tem mais como sair. Outra possibilidade seria fazer o computador ignorar a ordem.



Voltando ao original, precisaremos fazer a parte de baixo. Ela ficará quase igual a de cima.



Pronto! Alguma semelhança com a máquina do exemplo?

Agora, vamos pegar nossa sequência de **E**s e **D**s do começo da aula e ver que a MT funciona. Isso se deve porque ela é uma representação teórica de nosso computador do início da aula.

2.4 Relatório

A aula contou com a ajuda de dois voluntários: o Murilo e o César. Sem eles provavelmente teria sido bem mais difícil conversar com a turma.

Comecei a aula fazendo a chamada e perguntando aos alunos o que faziam na aula de Computação e o que queriam fazer na aula. Uns disseram que queriam aprender mais sobre Computação, outros disseram que adoram usar o computador e por isso estavam lá e, por fim, alguns fugiram da aula porque não teríamos computador naquela aula (portanto, nada de Facebook e joguinhos horrorosos em Flash).

Depois, começamos a Atividade: O Computador que Anda para Frente. O pessoal percebeu, não sei se influenciados por ser uma aula de Computação, que as instruções para fazer o se computador mover eram: esquerda e direita. Claro que apareceram coisas tais como *"Dê um mortal para trás!"* que o computador infelizmente não conseguiu interpretar.

Alguns alunos foram anotar na lousa o nosso "algoritmo" de movimentação do computador. Eles mesmos pediram para que os outros alunos falassem um de cada vez as instruções.

Visto, enfim, que o computador não passa de uma criatura burra, fomos investigar sua origem.

O segundo momento da aula foi um momento mais expositivo feito com base em A História Antiga da Computação.

Alguns deles se surpreenderam com o esquema do relógio. Outros prestaram mais atenção aos nomes diferentes que surgiram na aula.

Mostrei uma conta enorme para justificar os esforços para criarmos meios de fazer contas automaticamente, mas ninguém se animou a fazê-la até que um aluno perguntou se poderia usar o celular para resolvê-la.

Cabe salientar que no trecho expositivo a turma estava mais sossegada (suspeito que alguns deles estivessem bem longe da sala de aula). Mas até que a turma pareceu prestar um pouco de atenção.

Por fim, o último momento da aula. Quando tentamos transformar o computador que anda para frente numa Máquina de Turing.

Discutimos bem superficialmente sobre modelos teóricos e quis fazê-los construírem uma MT (claro que era praticamente um autômato, já que a "fita" só andava em uma direção e de uma em uma posição).

Na hora de montar a MT, bastou eu escrever o estado inicial e o $(0, 1)$ para logo sugerirem o que eu deveria fazer até construirmos o modelo.

Depois, simulamos alguns programinhas do Computador que Anda para Frente simultaneamente na MT e com alunos que foram até a frente.

Fiquei bastante surpreso que foi bem simples trabalharmos MTs, mesmo que de forma superficial.

Perguntei aos alunos, ao final da aula, o que acharam. Disseram que gostaram. (mas nunca se sabe!)

Em suma, a aula foi bem melhor do que eu imaginava!

Capítulo 3

Aula 2: Binários e *Bits*

Vende-se computador com dois gigas de memória e cem gigas de disco. Provavelmente, diversas ofertas que circulam em diversas mídias, tais como jornais, televisão, revistas, Internet..., acabam tendo um formato bastante parecido. Mas o que elas querem dizer? O que são esses *gigas*?

Para entendermos melhor o que querem dizer, precisaremos definir o que são números binários. A próxima atividade nos trará uma noção intuitiva.

Além disso é importante entender, a partir dos números binários, o conceito de *bit* e como ele aparece no computador.

3.1 Atividade: O Campeonato de Binário

3.1.1 Objetivos

Esta atividade tem como objetivos:

- Conhecer um novo sistema de representação numérica;
- Entender particularidades do sistema binário na prática.

3.1.2 Descrição

O jogo consiste em escrever a representação binária de um número a partir de cartas com pontinhos desenhados. Esta seção, em grande parte, é uma espécie de tradução de [8].

O número de pontos nas cartas se distribui a partir de 1 em potências de 2. Logo um jogo com k cartas terá cartas valendo: $1, 2, \ldots, 2^{k-1}$.

Nesta atividade, utilizamos um conjunto de seis cartas como ilustrado abaixo.

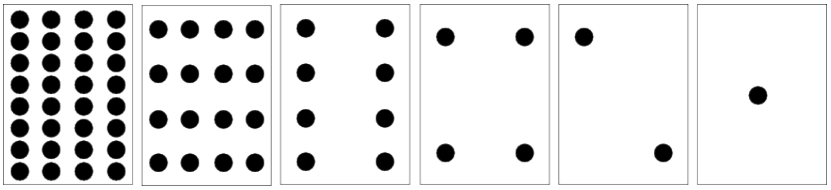


Figura 3.1: Cartas do jogo.

Para representar os números, é suficiente deixar para cima as cartas cuja quantidade de pontos for igual ao número escolhido e anotar 1 quando a carta estiver virada para cima e 0, caso contrário. Entretanto, vale ressaltar que nem todo número poderá ser representado com uma quantidade finita de cartas. Caso acontecer, anotar que não dá para representá-lo.

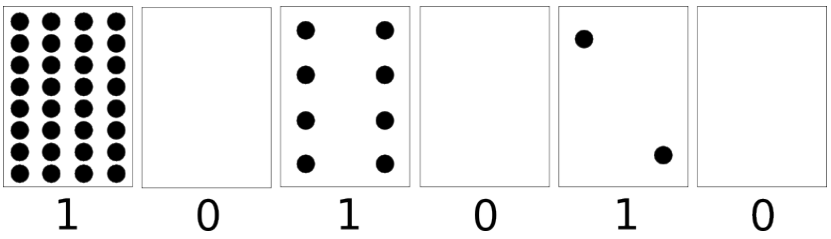


Figura 3.2: Número 42 escrito com as cartas do jogo.

Para o campeonato, recomenda-se dividir os alunos em grupos de 4 a 5. Uma vez divididos, distribuir os baralhos e explicar como montar os números com as cartas. É recomendado jogar umas três rodadas sem contar pontuação para verificar se os alunos aprenderam as regras.

Por fim, escolher com a turma uns 20 números e começar a jogar. Ganha o primeiro grupo que entregar anotadas as representações em binário, quando possível, para os números escolhidos.

3.2 Os Binários na Computação

Antes de contextualizarmos os binários na Computação, é importante entender o que é esse tal de sistema binário.

Os números, em geral, possuem diversas formas de representação. Podemos usar palitinhos, escrever os algarismos arábicos, utilizar algarismos romanos, escrever por extenso, escrever usando ideogramas etc. A partir da atividade anterior, vimos, também, que é possível representar números utilizando apenas *zeros* e *uns*.

> *Por que usar binário e não algarismos arábicos para trabalharmos com computadores?!*

Para responder a essa pergunta, precisaremos ver quantos símbolos temos em outros sistemas de representação. No decimal, por exemplo, usamos **dez** símbolos. Já no binário são apenas **dois**! Foi justamente por ter dois *estados*, com furo e sem furo, que o Tear de Jacquard desempenhou um papel importante na História da Computação.

Além disso, é bem simples identificar coisas que possuem dois estados, tais como: **ligado/-desligado**, **aberto/fechado**, **verdadeiro/falso** e **1/0**. Já identificar quão acesa está uma lâmpada não parece tão simples... Pior: um mecanismo poderia identificar que uma lâmpada parcialmente acesa como 10% acesa e outro mecanismo, como 50% acesa. Como garantir precisão?

A resposta encontrada na época, sobretudo pelo equipamento que tinham em mãos, foi justamente abandonar a ideia de trabalhar com mais de dois estados. Lembrando que os computadores são criaturas **simples**. Simples e burras.

Para ilustrar melhor a simplicidade do sistema binário em relação ao decimal, vamos ver como funciona a soma binária.

Número	Número	Soma
0	0	00
0	1	01
1	0	01
1	1	10

Como vimos na Atividade: O Campeonato de Binário, o número **00010** é o 2 e o número **00001**, o 1. Nada mais razoável que imaginar que $1 + 1 = 2$! Ou, em binário, que $1 + 1 = 10$.

Para completar o esquema da soma, precisamos, também, levar em consideração o *vai-um*.

Número	Número	Veio-um	Soma
0	0	0	00
0	1	0	01
1	0	0	01
1	1	0	10
0	0	1	01
0	1	1	10
1	0	1	10
1	1	1	11

A partir da tabela, conseguimos as seguintes possibilidades:

- $0 + 0 + 0 = 0$;
- $0 + 1 = 1 + 0 = 1$;
- $1 + 1 = 0$ e vai 1;
- $1 + 1 + 1 = 1$ e vai 1;

Ex:

$$\begin{array}{r} 100111 \\ +000011 \\ \hline 0 \end{array} \Rightarrow \begin{array}{r} 100111 \\ +000011 \\ \hline 10 \end{array} \Rightarrow \begin{array}{r} 100111 \\ +000011 \\ \hline 010 \end{array} \Rightarrow \begin{array}{r} 100111 \\ +000011 \\ \hline 1010 \end{array} \Rightarrow \begin{array}{r} 100111 \\ +000011 \\ \hline 01010 \end{array} \Rightarrow \begin{array}{r} 100111 \\ +000011 \\ \hline 101010 \end{array}$$

Legal! O computador trabalha com esse tal de binário. Mas como eles aparecem no computador? Qual a relação entre esse tal de binário e o Counter Strike que eu jogo?

Os binários aparecem no computador em forma de componentes que podem ser ligados e desligados a partir de operações. Antigamente, na época do ENIAC, usava-se válvulas para essa finalidade. Hoje temos os transistores cujo tamanho pode ser **bem** menor que um fio de cabelo!

Associando esses componentes, podemos representar uma quantidade enorme de coisas. Se dissermos ligado **L** vale 1 e desligado **D** vale 0, teremos algo com as mesmas propriedades do sistema binário a partir desses componentes.

Ex: **LDLDLD** pode ser visto como **101010**.

Bit veio do inglês *binary digit* que significa dígito binário. Quando associamos oito *bits*, obtemos um *byte*.

Voltando aos *gigas*, geralmente o que aparece de forma mais precisa é *gigabytes*. O termo *giga* significa "um bilhão". Logo quando vemos a propaganda dizendo 2 *gigabytes* de memória, significa que o computador conseguirá guardar cerca de **dois bilhões** de *bytes* na memória. Dois bilhões de zeros e uns!

Para chegarmos no jogo, precisaremos de diversas coisas. A primeira delas é uma forma de transformar esses números e será vista na próxima aula!

3.3 Relatório

A aula contou com a presença de dois voluntários: o André e o Gustavo. O André aproveitou para apontar detalhes que poderiam ser consertados e o Gustavo participou da aula.

Comecei a aula fazendo a chamada e perguntando aos alunos o que faziam na aula de Computação e o que acharam da aula anterior. Uns disseram que gostaram, outros disseram que foi engraçado e, por fim, outros não haviam vindo na aula anterior.

Depois, começamos a Atividade: O Campeonato de Binário. Pedi para o pessoal se separar em grupos. Na realidade, eu determinei quais seriam os grupos a partir da posição das pessoas na sala de aula. Houve, em princípio, um certo choro para que eu mantivesse as panelinhas, mas *nada de panelinha*.

Expliquei as regras do jogo para o pessoal - que não entendeu muito bem no começo. Mas, depois de uns exemplos, logo perceberam. Acabei, também, dizendo para escreverem **C** para carta virada para cima e **B** para carta virada para baixo. Talvez devesse ter começado direto com os **1s** e **0s**.

Depois de umas três rodadas, já tinham desvendado o algoritmo para descobrir a representação binária dos números. O algoritmo era:

1. Subtrair o maior número das cartinhas que *cabe* no número que a turma escolheu;

2. Caso o número que sobrou for diferente de zero, repetir.

Dez minutos depois e toda a turma já havia terminado. Um dos grupos terminou quase instantaneamente. Acabei propondo o seguinte problema: "Existe algum número que não possamos representar com as cartinhas?"

Para minha surpresa, disseram que qualquer número que fosse maior que 63 não poderia ser escrito. Depois de um tempinho, concluíram que qualquer número de 0 a 63 poderia ser representado pelo conjunto de cartinhas.

Terminado o jogo, pedi para que os alunos fossem na lousa escrever as respostas. Cada um acabou indo pelo menos uma vez. Após devidamente corrigirmos a atividade, fiz a mesma pergunta do desafio para a turma. Acabamos concluindo que, com uma quantia suficiente de cartinhas, poderíamos representar qualquer número natural.

Findada a discussão, começamos Os Binários na Computação.

Diferente da seção expositiva da aula anterior, as pessoas se dispersaram menos. Entretanto, não foi tão simples convencê-las a experimentar as contas em binário. Por sorte, um aluno tentou e o pessoal viu que era trivial.

Pedi para que todos os alunos fizessem, na lousa, ao menos uma continha. Alguns alunos disseram que era muito fácil, mas se desanimaram ao verem contas com dez algarismos.

O trágico final da aula se deu quando tentamos ver um pouco de lógica. Meu objetivo era ver tanto os bits quanto as portas lógicas na mesma aula... Um objetivo não muito inteligente, diga-se.

Acabamos vendo o conceito de verdadeiro, falso e os conectivos. Mas quase ninguém entendeu nada e a maioria da turma ficou boiando... Ficou a lição: **nunca** começar um assunto novo nos vinte últimos minutos de aula!

Justamente por isso acabei, no material, riscando o trecho que se referia à lógica.

Capítulo 4

Aula 3: Lógica e Portas

Até o presente momento, vimos que computadores são criaturas burras descendentes das calculadoras. Também vimos que as coisas num computador são representadas pelos números binários. Entretanto, ainda não vimos como esses binários são transformados.

Para entender melhor como funcionam essas transformações, precisamos, antes, ver um pouco de Lógica. A próxima atividade e o próximo assunto serão relacionados à Lógica.

Numa outra seção, veremos o que são portas e tentaremos construir um circuito a partir delas.

4.1 Atividade: Detetive

4.1.1 Objetivos

Esta atividade tem como objetivos:

- Exercitar o raciocínio lógico;
- Conhecer os operadores lógicos de um jeito prático.

4.1.2 Descrição

Tanto o professor quanto os alunos são detetives de uma agência. Detetives que no momento não puderam coletar pistas sobre casos, mas ainda podem deduzir coisas importantíssimas para resolvê-los.

Há diversos tipos de clientes numa agência de detetives, entretanto, na aula, serão resolvidos apenas dois casos:

- Caso 1: a melhor banda de todos os tempos da última semana;
- Caso 2: reforços para o Brasileirão.

4.2 Um pouco de Lógica

Por diversas vezes, na atividade anterior, acabamos nos deparando com deduções que associavam duas informações para obtermos uma nova. Ex:

> Bernard tocou guitarra ou bateria.
> Bernard não tocou guitarra.

Se ele tocou guitarra ou bateria e não tocou guitarra, então só pode ter tocado bateria.

Por outro lado, utilizamos as dicas dos detetives sem pensar muito se eram, de fato, verdadeiras. Por sorte, esse tipo de problema não nos prega peças a partir de informações falsas.

O objetivo da Lógica, de certa forma, é conseguir deduzir novas sentenças a partir de sentenças pré-existentes levando principalmente em consideração se essas sentenças são verdadeiras ou não.

Para deduzirmos novas sentenças empregamos os *conectivos*. De certa forma, já fizemos isso ao associar duas dicas e obter uma informação enquanto éramos detetives. O conectivo que produz uma sentença a partir de duas é o e.

Mas como funciona esse e?

A ideia é bem simples. Se tivermos duas sentenças que são simultaneamente verdadeiras, então o e das duas é verdadeiro. Caso contrário, é falso. A partir dessas informações, conseguimos construir a seguinte *tabela verdade*:

Sentença 1	Sentença 2	S1 e S2
F	F	F
F	V	F
V	F	F
V	V	V

No jogo de detetive, também nos deparamos com o não. Ele funciona justamente como imaginamos: se algo é verdadeiro, o não algo é falso.

Exemplo relacionado ao Caso 2: reforços para o Brasileirão:

O holandês jogará no Corinthians. (verdadeiro)
O holandês não jogará no Corinthians. (falso)

A tabela verdade do não é bem mais simples.

Sentença 1	não S1
F	V
V	F

Por fim, falta vermos o ou, aquele mesmo que apareceu no Caso 1: a melhor banda de todos os tempos da última semana.

Se de duas sentenças pelo menos uma delas for verdadeira, então o ou delas é verdadeiro.

Exemplos:

O teto é para cima ou o chão é para cima.

É uma sentença que faz sentido, pois o teto é para cima[1].

Mas

O teto é para baixo ou o chão é para cima.

Não faz tanto sentido. Nem o teto é para baixo, nem o chão é para cima.

Já

O teto é para cima ou o chão é para baixo.

Também faz sentido. Se ambas sentenças forem verdadeiras, uma delas necessariamente é verdadeira.

Isso nos leva à seguinte tabela:

[1] Assim eu espero!

Sentença 1	Sentença 2	S1 ou S2
F	F	F
F	V	V
V	F	V
V	V	V

Mas qual a relação entre essa tal de Lógica e as transformações de binários?

4.3 Portas Lógicas

Vimos na aula anterior como funciona a soma de números binários, mas será que o computador também faz dessa forma?

É possível que exista algum circuito que trabalhe a soma como vimos, contudo o que mais encontramos são circuitos baseados em *portas lógicas*.

Uma porta lógica é um componente que, a partir de um ou mais bits, consegue devolver outro bit. Ela funciona como uma máquina de transformar bits. O nome lógica vem justamente da Lógica que acabamos de ver.

Imagine que trocamos os verdadeiros e falsos por 0s e uns 1s da seguinte forma:

- $F \mapsto 0$;
- $V \mapsto 1$.

Para cada conectivo da seção anterior temos uma porta lógica cujo comportamento é idêntico ao descrito pela tabela verdade.

Número	Número	N1 e N2
0	0	0
0	1	0
1	0	0
1	1	1

Legal. Agora temos portas lógicas e binários. Mas como utilizar essas portas?

Vamos tentar construir a soma binária a partir dessas portas. Mas, antes, precisaremos de uma porta lógica que ainda não vimos, a xou.

Seu funcionamento é *quase* igual ao da porta ou, mas ela nega V ou V.

Número	Número	N1 xou N2
0	0	0
0	1	1
1	0	1
1	**1**	**0**

Sejam n_1 e n_2 os dois números que queremos somar. Vamos olhar para a soma que vimos na aula passada, mas de uma forma diferente. Vamos olhar somente para o dígito da direita.

n_1	n_2	$n_1 + n_2$
0	0	$0\underline{0}$
0	1	$0\underline{1}$
1	0	$0\underline{1}$
1	1	$1\underline{0}$

Olhando um pouco para cima, percebemos que é **exatamente** o mesmo comportamento de quando usamos a porta XOU! Será que existe alguma porta cujo comportamento é parecido com o dígito da esquerda?

n_1	n_2	$n_1 + n_2$
0	0	00
0	1	01
1	0	01
1	1	10

A resposta é sim. Basta verificarmos como se comporta a porta E. Portanto, a partir de uma porta do tipo XOU e uma do tipo E conseguimos construir um circuitinho de soma!

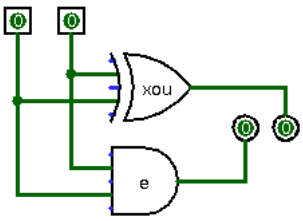


Figura 4.1: 0+0 = 00

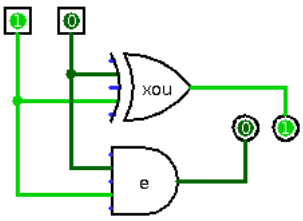


Figura 4.2: 1+0 = 01

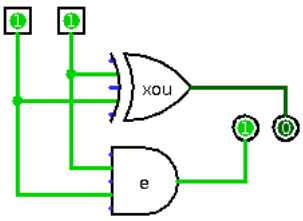


Figura 4.3: 1+1 = 10

Podemos encontrar um exemplo de como construir esses circuitinhos aqui:

`http://www.youtube.com/watch?v=Zkkck2MovCc`

Agora vimos que o computador trabalha com binários e consegue transformá-los. Para finalmente chegarmos nos programas, falta construirmos a ponte entre o binário e o que nós, seres humanos, passamos para o computador.

Continua no próximo episódio!

4.4 Relatório

Comecei a aula fazendo a chamada e perguntando aos alunos o que faziam na aula de Computação e o que acharam da aula anterior. Uns disseram que gostaram, outros disseram que foi mais ou menos. Provavelmente por causa da grande confusão da aula anterior.

Depois, revisamos rapidamente o que foi visto até o presente momento. Devido às faltas, diversos alunos desconheciam diversas coisas...

A sequência da aula se deu com a Atividade: Detetive. Pedi para que dispusessem as carteiras num formato de U e, depois, contei que, agora, éramos detetives e precisávamos resolver dois casos.

O desenrolar da atividade foi bastante divertido. Alguns alunos sabiam claramente qual a resposta, mas se embananavam na hora de explicar. Outros conseguiram explicar com primazia o raciocínio que desenvolveram para solucionar os problemas.

Em ordem cronológica, comecei com o Caso 2: reforços para o Brasileirão e, em seguida, partimos para o Caso 1: a melhor banda de todos os tempos da última semana. Tomando cuidado para deixar as anotações em lousa devidamente diagramadas, uma vez que eu dependia das informações para a parte posterior a aula.

Diferentemente da aula anterior, começar com um jogo de lógica ajudou bastante na hora de trabalharmos os conceitos de lógica formal. Além disso, o uso de exemplos concretos - tal como sugerido pelo voluntário André depois da aula anterior -, permitiu uma abordagem bem mais simples.

Conseguimos construir a tabela do e e a do não tranquilamente. O conceito de ou já sofreu uma resistência um tanto maior.

Depois, continuamos com as portas lógicas. Como o tempo estava acabando, ficou um tanto corrido. Contudo o pessoal conseguiu entender que era possível transformar bits no computador. Não tenho certeza se entenderam exatamente como funcionam as portas... De qualquer forma, não é da alçada do curso ser tão específico.

A partir da quarta semana, os alunos decidem em qual atividade seguirão no restante do semestre. Tendo isso em vista, parei a aula um pouco mais cedo para decidirmos o que fazer a partir da outra semana.

A primeira possibilidade, seria montar o circuitinho de soma visto em aula e generalizá-lo. Já a segunda, começar finalmente a desbravar o conteúdo de programação. Quase unanimemente a turma optou pela programação.

Também ao final da aula, propus o Desafio: O Teste de "Einstein". Apenas um aluno não quis ficar com as folhas do desafio quiçá tentar resolvê-lo. Espero que alguém consiga ou, ao menos, venha falar comigo.

4.5 Caso 1: a melhor banda de todos os tempos da última semana

4.5.1 Histórico

Um power trio é um conjunto musical que possui três músicos.

George, Theófilo e Bernard têm um power trio, mas o fato curioso é que todos eles podem tocar qualquer instrumento dentre estes: bateria, guitarra e teclado.

Um fã nos contratou para descobrirmos quem tocará o quê e onde ficarão no palco na próxima apresentação. Enviamos dois detetives para observar os ensaios e descobrir a preciosa informação para nosso cliente... Mas nossos detetives escolheram um local ruim e cada um acabou observando o ensaio a partir de uma das janelas. `u_U"`

4.5.2 Enunciado

O fã desvairado, ao solicitar nossos serviços, trouxe as seguintes informações:

- Há três membros na banda: George, Theófilo e Bernard;
- No ensaio, cada músico usa somente um instrumento e não há repetição;
  Ex: dois caras não podem tocar guitarra no mesmo ensaio.
- Os músicos não trocam de lugar durante o ensaio. Além disso, o lugar do ensaio será o lugar onde ficarão no show.

4.5.3 Informações

Depois de um árduo trabalho, nossos colegas detetives trouxeram as seguintes informações:

1. O detetive 1 disse que o Bernard tocou guitarra ou bateria;
2. O detetive 2 disse que o Bernard não tocou guitarra;
3. O detetive 1 disse que o George não tocou teclado;
4. O detetive 2 disse que o tecladista ficou à direita;
5. O detetive 1 disse que o guitarrista, que certamente não era o Theófilo, ficou à esquerda.

4.6 Caso 2: reforços para o Brasileirão

4.6.1 Histórico

Para disputar o Brasileirão de 2013, três gringos foram contratados para atuar em três times brasileiros. No entanto, a imprensa não conseguiu descobrir onde irão atuar e de que país eles vieram.

Logo vieram nos contratar para que possamos descobrir essa informação o **mais rápido possível**. Será o furo de reportagem do ano!

Os jogadores são:

- Cristiano Ruinaldo;
- Wayne Ruiney;
- Román Ruinquelme.

4.6.2 Informações

Nossos bravos detetives, depois de virarem noites observando a rotina dos atletas, obtiveram as seguintes informações:

1. Cada jogador atua somente por um clube;
2. Os clubes são: Corinthians[2], Palmeiras e São Paulo;
3. As nacionalidades: argentino, francês e holandês;
4. O holandês jogará no Corinthians;
5. Ruinquelme é argentino;
6. Ruiney não jogará no Palmeiras;
7. Cristiano Ruinaldo não é francês.

[2]Atual campeão mundial!

4.7 Desafio: O Teste de "Einstein"

4.7.1 Histórico

Reza a lenda que Albert Einstein escreveu um teste de lógica para se livrar de seus possíveis orientandos. Também reza a lenda que 98% da população mundial não consegue resolver tal teste.

Aparentemente, afora o folclore, não há registro que o teste seja realmente de autoria do Einstein, entretanto, é um excelente desafio!

4.7.2 Enunciado

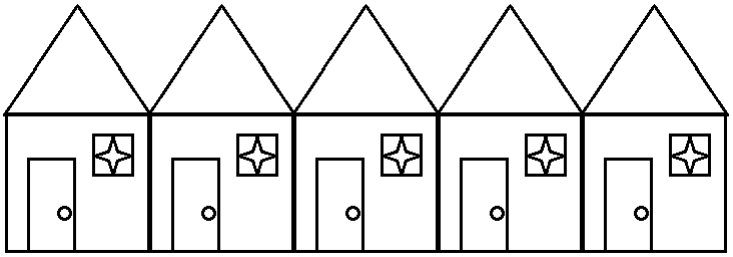


Figura 4.4: Disposição das casas.

- Há cinco casas de diferentes cores que ficam na mesma calçada;
- Em cada casa vive uma pessoa de um país diferente;
- Cada um desses caras:
  - gosta de uma bebida;
  - gosta de uma marca de cigarro;
  - tem um animal de estimação.
- Não tem bebida, marca de cigarro ou animal repetido.
  Ex: dois caras não podem ter cachorro como animal de estimação.

A grande questão é:

"Quem deles tem um peixe?"

4.7.3 Informações

1. O **INGLÊS** mora na casa **VERMELHA**;
2. O **SUECO** tem um **CACHORRO**;
3. O **DINAMARQUÊS** bebe **CHÁ**;
4. A casa **VERDE** fica do lado **ESQUERDO** da casa **BRANCA**;
5. O dono da casa **VERDE** bebe **CAFÉ**;
6. O rapaz que gosta da marca **PALLMALL** cria **PÁSSAROS**;
7. O dono da casa **AMARELA** gosta da marca **DUNHILL**;
8. O dono da casa do **MEIO** bebe **LEITE**;
9. O **NORUEGUÊS** mora na **PRIMEIRA** casa (a mais da esquerda);
10. O rapaz que gosta da marca **BLEND** tem um **VIZINHO** que tem um **GATO**;
11. O homem que gosta da marca **BLUEMASTER** bebe **CERVEJA**;
12. O rapaz que tem um **CAVALO** mora **DO LADO** do que gosta de **DUNHILL**;
13. O **ALEMÃO** gosta da marca **PRINCE**;
14. O **NORUEGUÊS** mora **DO LADO** da casa **AZUL**;
15. O rapaz que gosta da marca **BLEND** tem um **VIZINHO** que bebe **ÁGUA**.

Capítulo 5

Aula 4: Códigos e Criptografia

Durante nossa jornada pelo mundo dos uns e zeros, vimos, na aula passada, que é possível transformar esses uns e zeros em **outros** uns e zeros. Se por um lado isso explica o funcionamento de coisas como as contas num computador, por outro, não explica como este texto foi redigido ou como as imagens são representadas. Principalmente, não explica qual a relação dos uns e zeros com essas coisas.

O objetivo da aula de hoje é justamente trabalhar esses conceitos. Primeiro vamos conhecer o que é criptografia e, depois, codificação de um jeito mais geral.

5.1 Atividade: Criptoanálise

5.1.1 Objetivos

Os objetivos desta atividade são:

- Trabalhar criatividade e investigação;
- Introduzir o conceito de código;
- Ilustrar o conceito de ciclo.

5.1.2 Descrição

Tentar decifrar o texto abaixo junto com a turma. Recomenda-se escrevê-lo na lousa para que a turma toda participe.

Anevm, anevm, r anevm

Anevm, anevm, r anevm,
Anevm, dhr ahapn fr npnon;
Anevm, dhr fr ryr qrfnon,
Snen b zhaqb vasryvm;
Anevm, dhr Arjgba anb dhvf
Qrfperire-yur n qvntbany;
Anevm qr znffn vasreany,
Dhr, fr b pnyphyb anb reen,
Cbfgb rager b Fby r n Green,
Snevn rpyvcfr gbgny!

Dicas

- O texto está em Português exceto pelas letras, que foram trocadas;
- Não há diferença entre as letras maiúsculas e minúsculas;
- Não há acentuação na versão codificada;
- $A = N$ e $M = Z$.

5.2 Cifra de César

5.2.1 Gaius Julius Caesar

Gaius Julius Caesar, nascido em 100 A.C. e morto em 44 A.C., foi um dos responsáveis pelo que veio a ser conhecido como Império Romano. Durante seu período no poder o ditador, outrora general, conquistou inúmeros territórios.

Mas nem só de militarismo viveu *Júlio César*. Antes da vida militar, ele atuou, dentre outras coisas[1], como promotor, sendo responsável pela condenação de ex-governantes notórios por corrupção [5]. Porém, por querer concentrar poder, Júlio César morreu vítima de uma conspiração feita pelo senado romano.

5.2.2 A cifra

Júlio César, tanto o general quanto o estadista, tinha ideia que era preciso evitar que qualquer informação vital caísse em mãos inimigas. Entretanto, nem sempre ele poderia entregar essas informações pessoalmente. A solução foi justamente *cifrar* as correspondências que enviava através de seus emissários e evitar vazamento de informação caso alguém obtivesse acesso ao texto.

A estratégia era simplesmente **andar três letras para frente**.
Exemplo:

YDL FRULQWKLDQV![2]

Por outro lado, não é suficiente *apenas* andar três letras para frente. Não existe letra que venha depois de "Z", por exemplo. Se imaginarmos que "A" vem depois de "Z", ou seja, dermos a **volta** no alfabeto, cada letra terá uma letra correspondente. Além disso, desfazer é bem simples. Basta voltar cada uma das letras três posições lembrando que "Z" virá *antes* de "A".

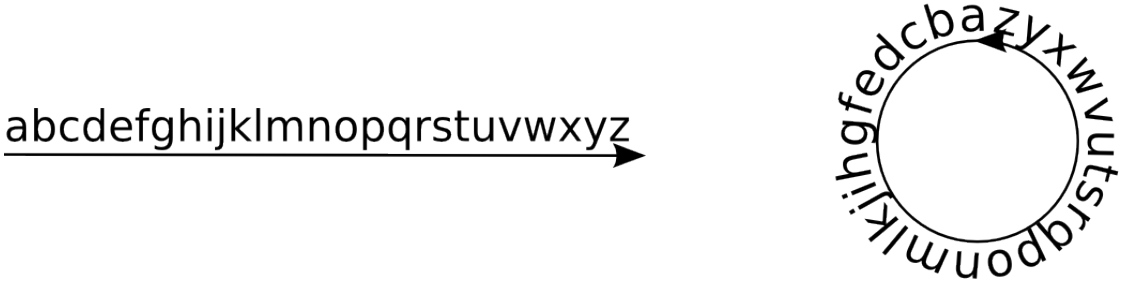


Figura 5.1: Sequência comum do alfabeto e alfabeto "dando a volta".

Embora simples, a Cifra de César pareceu funcionar muito bem na época.

[1] Tal como sacerdote de Júpiter!
[2] VAI CORINTHIANS!

5.2.3 Como quebrar a cifra?

Na época de César, época esta onde a maioria das pessoas sequer sabia ler, a cifra funcionou. Mas o que poderíamos fazer para quebrá-la? Conhecendo a estratégia utilizada para sua construção, há um recurso bastante simples para descobrirmos a chave: **força bruta**.

Um adversário de César poderia, por exemplo, contratar 23^3 pessoas que saibam ler e mandar cada uma decodificar da seguinte forma: a primeira anda apenas uma casa para trás, a segunda, duas casas...

Caso as palavras estivessem agrupadas tal como no Latim, poderíamos procurar por palavras que aparecem com bastante frequência. Também poderíamos proceder da mesma forma para as letras, afinal, cada idioma tem alguma letra cuja frequência é maior. No Português, por exemplo, a letra é **A**. Essa técnica é conhecida por **Análise de Frequência**.

5.3 Códigos

A Cifra de César é um exemplo de código, pois temos a informação (texto cifrado) e o jeito de interpretá-la (chave). Porém, nem todo código foi feito para ser secreto.

Sistemas de codificação têm diversas utilidades quando precisamos trabalhar com diferentes tipos de meios como eletricidade, som, radiofrequência etc. Esses sistemas aparecem em muitos lugares como:

- O sinal que o controle remoto envia para a televisão ao apertarmos o botão. Caso não houvesse um código, qualquer controle poderia ligar qualquer televisão ou a televisão sequer ligaria porque o sinal não seria intepretado;
- A tecla no computador quando apertada envia os nossos amigos zeros e uns no formato de sinais elétricos para o computador. O computador, por sua vez, reconhece esses sinais e os transmite para alguma coisa coisa como um jogo, uma ferramenta de escrever texto etc;
- Alguns computadores, quando ligados, emitem um sinal sonoro que muda caso tenha acontecido alguma coisa de ruim.

Num computador existem diversos sistemas de codificação. Um exemplo é justamente o sistema que transforma os uns e zeros em **letras**. Há inúmeros sistemas de codificação de sequências de *bits* para letras, mas, na aula de hoje, veremos apenas o `ASCII`

5.3.1 ASCII

American Standard Code for Information Interchange ou Código Padrão Americano para Troca de Informações é um código que associa 7 *bits* a símbolos como letras e números.
Exemplo:

`1000001` representa a letra 'A' no ASCII.

Para saber como uma letra é representada no sistema ASCII, podemos usar as seguintes regras (onde o primeiro *bit* será mais da esquerda):

bit 1	*bit* 2	Tipo
1	0	letra maiúscula
1	1	letra minúscula
0	1	número

[3] O alfabeto latim da época do Império Romano tem 23 letras. Lá não aparece as letras: **J**, **U** e **W**.

A partir da tabela, podemos enumerar as letras assim: $A = 1$, $B = 2$, ..., $Z = 26$. Daí é só juntar os dois primeiros *bits* como os da tabela com o número binário correspondente. Já para os números, podemos proceder de maneira similar: $0 = 0$, $1 = 1$, ..., $9 = 9$.
Exemplos:

1. `1000100` é "D", já que 4 em binário é 01000;
2. `1100010` é "b", pois 2 em binário é 00010;
3. `1101011` é "k", já que 11 em binário é 01011;
4. `0110011` é "3", pois 3 em binário é 011.

Fazendo umas contas, percebemos que o máximo de símbolos que poderíamos representar utilizando 7 *bits* é 127. Por outro lado, os números de 0 a 9, as letras minúsculas e as letras maiúsculas correspondem a 62 símbolos. É importante perceber que nem só de letras e números escrevemos texto. Também utilizamos diversos símbolos como pontuação, hífen, parênteses etc. Todos esses símbolos também têm sua representação em sete *bits*.

Contudo, mesmo com 127 símbolos não conseguimos, por exemplo, representar todas as letras incluindo símbolos para cada uma das versões acentuadas de letras.

5.4 Relatório

Antes de escrever qualquer coisa, é importante ressaltar que o conteúdo da aula 4 foi planejado às pressas devido a um problema com o laboratório.

A aula contou com a presença do César que cuidou, inclusive, de fazer a chamada.

Comecei a aula escrevendo a poesia do Bocage na lousa dando início à Atividade: Criptoanálise. Perguntei aos alunos se entendiam a minha letra e, principalmente, se entenderam o que estava escrito na lousa.

Disseram que a letra estava legível, mas não faziam ideia do que eram aquelas palavras tais como "anevm". Disse-lhes que o objetivo da aula era justamente descobrir o que estava escrito.

Um aluno perguntou se o texto na lousa estava em Português. Respondi, mas mesmo assim ninguém tinha ideia de como começar a decifração. Dei duas dicas:

1. A = N;
2. M = Z.

Depois da dica, tudo pareceu mais fácil. Um aluno disse que a palavra "anevm" era "nariz". O pessoal concordou e começamos a trocar as letras. Daí em diante anotamos na lousa as igualdades e foi questão de alguns minutos para termos o texto todo decifrado. Um fato curioso foi a palavra "Newton" que, em princípio, gerou uma certa dúvida.

Finda a atividade, reordenei o mapa na lousa de forma como a tabela abaixo:

A	B	C	D	E	F	G	H	I	J	K	L	M	N
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A

Enquanto escrevia a tabela ouvi um "não acredito!" em tom de risada. Alguns alunos perceberam que a resposta estava exatamente abaixo de seus "anevmrf".

Terminada a tabela, começamos a explicação sobre a Cifra de César. Vimos o conceito de chave e qual era a chave para a tabelinha do problema do começo da aula. Perguntei se havia alguma chave que "andar para frente" é igual a "andar para trás". Logo um aluno respondeu: "26! 26!!!". Sempre a resposta trivial...

Tive, depois, de refazer a pergunta excluindo qualquer múltiplo de 26. Passou um tempinho e recebi um 39 como resposta. Ok! Uma resposta válida. Depois, chegamos à conclusão que 13 também serve. Foi justamente por isso que eu escolhi 13 como chave.

Explicada a cifra, desde sua criação até sua aplicação, começamos a discussão sobre como quebrá-la. Visto os pontos-fracos da cifra, perguntei aos alunos se eles tinham uma ideia de como melhorá-la e, para minha grata surpresa, um deles sugeriu que andássemos quantidades diferentes por letras. Ele disse algo como: "Faz assim: anda uma casa, depois anda duas, depois anda três e por aí vai.". Ele também disse que havia visto um filme sobre *hackers* e criptografia.

Terminado o trecho de criptografia, partimos para a seção de Códigos. Tentamos, então, definir o que é código a partir da ideia da criptografia e localizar onde mais eles aparecem. Depois de uns exemplos, concluímos que existem códigos que servem para esconder coisas, mas, existem, também, códigos para traduzir coisas.

Contei sobre a tabela `ASCII`, listei as letras "A", "B" e "C" e perguntei se alguém sabia como seria a letra "P". Alguns alunos acertaram, outros não quiseram responder.

No final da aula, sentamos para decidir o que seria melhor vermos na próxima aula que deveria ser no laboratório. Deixei duas possibilidades para decidirem:

1. Construímos o circuitinho de soma a partir das portas que vimos na aula anterior; ou
2. Acabamos com isso e começamos, de fato, com a programação.

Por quase unanimidade o pessoal decidiu começar com a programação.

Capítulo 6

Aula 5: Introdução ao *Python*

Nos episódios anteriores, conhecemos a "burrice" dos computadores e o sistema binário. Também estudamos como transformar esses binários e um pouquinho de codificações.

Agora, precisamos ver como dar ordens ao computador e precisamos, também, entender exatamente como é programar um computador. Para tanto, vamos trabalhar com uma linguagem de programação chamada *Python*.

A escolha de Python se deve a algumas características da linguagem que serão vistas adiante.

6.1 Python

Nos final dos anos 80 e começo dos anos 90, Guido van Rossum foi o principal por responsável especificar e dar a vida à linguagem de programação que conhecemos como *Python*.

Ela é uma linguagem interativa, portanto é possível[1] escrever e rodar o código linha por linha. Essa característica não é valida em todas as linguagens de programação.

Além disso, a linguagem foi projetada de forma a tentar deixar o código o mais legível possível. Futuramente veremos o que é um código legível.

Outra vantagem de usar Python é o fato de conseguirmos trabalhar com diversos dispositivos desde computadores rodando Windows, Linux e Mac até mesmo celulares!

De onde vem o tal do nome?

Procurando no dicionário, observamos que a palavra *python* é o nome, em inglês, de uma cobra enorme, a píton!

Contudo, não é por causa da cobra que a linguagem foi batizada assim. O nome Python da série de TV "*Monty Python's Flying Circus*".

Chega de *blablablá* e vamos ver como funciona a linguagem!

6.2 Interpretador

Quando abrimos o "Python IDLE", deparamo-nos com uma janelinha simpática onde está escrito alguma coisa parecida com o abaixo.

```
Python 2.7.3 (default, Jan  2 2013, 16:53:07)
[GCC 4.7.2] on linux2
Type "copyright", "credits" or "license()" for more information.
>>>
```

Chamamos essa janelinha simpática de *prompt* ou interpretador. É nela onde trabalharemos principalmente na aula de hoje.

[1] Apesar de não recomendável!



Figura 6.1: Píton. Obtida em: `http://commons.wikimedia.org/wiki/User:Bcsr4ever`

6.3 Primeiro Contato

Finalmente desbravaremos a linguagem e poremos a mão na massa. Para tanto, precisaremos reconhecer alguns recursos da linguagem.

6.3.1 *print*

Historicamente a primeira coisa que as pessoas costumam ver ao estudarem qualquer linguagem de programação é como escrever na tela. Daqui em diante, utilizaremos a palavra *imprimir* em vez da palavra escrever.

Uma das possíveis razões pelas quais as pessoas costumam aprender primeiro como escrever na tela é obterem uma resposta simples e direta do computador. Se houver erro na instrução de escrever na tela, então muito provavelmente ou o computador não escreverá nada ou escreverá alguma coisa bastante diferente do que programamos.

Quando tivermos um exemplo de janelinha com o *prompt*, digitaremos o que está na frente do `>>>`. Logo abaixo, aparecerá o resultado. Vamos tentar!

```
>>>print("Ai jow! É meu primeiro código! :D")
Ai jow! É meu primeiro código! :D
```

Simples assim! Qualquer coisa que estiver entre as aspas[2] será impressa na tela.

```
>>>print("POSSO ESCREVER O QUE QUISER!!!")
POSSO ESCREVER O QUE QUISER!!!
```

Nem só de impressão na tela são feitos programas de computador. Vejamos mais coisas que poderemos fazer com o Python.

6.3.2 Operadores

No contexto de Python, operadores não são os rapazes que operam máquinas em fábricas. Também não são os médicos que fazem cirurgia[3]. O nome *operador* vem justamente das operações matemáticas vistas com diversos detalhes no curso de Matemática: adição, subtração, multiplicação etc.

[2]Será que é qualquer coisa mesmo?

[3]Esses são os cirurgiões.

Como visto nas aulas anteriores, os computadores tiveram sua criação bastante ligada à necessidade de fazer contas de um jeito rápido e preciso. Faz sentido as linguagens nos fornecerem esse tipo de estrutura.

Vamos experimentar!

```
>>>2+2
4
>>>3-2
1
```

Parece funcionar!

```
>>>2x3
SyntaxError: invalid syntax
```

Não funcionou! O que aconteceu? Não era para ser uma multiplicação?

O símbolo x não é utilizado para multiplicação em Python. Se quisermos fazer multiplicações, precisaremos usar o símbolo $*$, o *asterisco*.

```
>>>3*2
6
>>>4/2
2
>>>4/3
1
```

Se quisermos trabalhar com números "decimais", utilizamos o . (ponto) em vez da vírgula.

```
>>>2*3.1415926535*4
25.132741228
>>>4.0/3
1.3333333333333333
```

A título de curiosidade, o Python também fornece operadores para resto de divisão inteira e até para potência! Chamamos o resto de divisão de *módulo*.

```
>>>4%3
1
>>>2**4
16
```

Mesmo assim, ele não faz milagres. Não conseguiremos, por exemplo, dividir por zero.

```
>>>4/0

Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    4/0
ZeroDivisionError: integer division or modulo by zero
```

Outro tipo de milagre que o computador não faz é conseguir perceber algumas contas parecidas com *"Quanto é dois mais dois dividido por três?"*. Puxa! Mas é $2 + 2$ e depois dividimos por 3? Ou é 2 somado com $2/3$?

O resultado da primeira conta é 1, já o da segunda, 2. E agora? Como fazer para dizer exatamente que conta queremos para o computador?

Podemos usar *parênteses* () igual já fazemos na Matemática.

```
>>>(2+2)/3
1
>>>2+(2/3)
2
```

A tabela abaixo contém um resumo dos operadores que encontramos em Python.

Operação	Operador
Adição	$+$
Subtração	$-$
Multiplicação	$*$
Divisão	$/$
Potênciação	$**$
Módulo	$\%$

6.4 Problemas

1. Um paladino aspirante tem 2147 peças de ouro e precisa comprar o máximo possível de poções médias e quer o troco em pergaminhos de portal.

   As poções custam 71 peças de ouro e servem para restaurar energia. Já os pergaminhos, 17 peças de ouro. A utilidade do pergaminho é escapar imediatamente do labirinto evitando o encontro **desagradável** com ogros, zumbis etc.

   Use o que aprendemos com o Python para resolver esse problema!

2. Você conhece alguma utilidade para o módulo[4]?

[4]Resto de divisão!

6.5 Relatório

Primeira aula no laboratório. Contei com a ajuda do César que acabou, também, fazendo a chamada. Também participou da aula o João Vitor que ajudou bastante com as atividades. A aula começou um pouco atrasada, tive de arrumar umas configuraçõezinhas nos computadores - tal como tirar o DNS das máquinas -, mas acabei indo para o laboratório meio tarde.

Alguns alunos começaram a explorar o que o sistema tinha a lhes oferecer. Uns abriram o LibreOffice, outros o GIMP, outros o terminal... mas o que todos queriam felizmente não estava disponível disponível: a Internet.

Comecei a aula contando-lhes um pouco sobre Python, também aproveitei para falar um pouco sobre Linux. Disse de onde veio o Python, comentei sobre a píton e logo pusemos a mão na massa. Abrimos o IDLE[5], escrevi a instrução "`print("AEAEAE! Consigo escrever!")`" e pedi para que copiassem. Logo um aluno perguntou se precisava copiar igual ou poderia trocar o que estava entre aspas. Aproveitei o ensejo para sugerir que escrevessem o que achassem mais divertido.

Depois do *print*, sugeri que fizéssemos umas continhas, afinal, foi para isso que o computador surgiu. Escrevi no quadro diversas contas e tentamos deduzir a partir dos resultados que operações cada símbolo representava. Inclusive, tentamos fazer contas que são impossíveis[6]. Até o momento descrito no parágrafo, os alunos, a menos de erros de sintaxe tais como esquecerem das aspas ou espaçamento, conseguiram levar as atividades sem problemas.

Dado que conseguíamos fazer contas utilizando o computador pareceu razoável fazermos um exercício, o primeiro da seção Problemas. Contei a historinha do paladino para os alunos e dei um tempinho para que fizessem as contas. Diferente das continhas simples, o problema ofereceu alguma dificuldade aos alunos. Entretanto, alguns conseguiram fazê-lo, inclusive usando o IDLE, sem problemas.

Terminamos a aula discutindo sobre como seria ruim ficar mudando o valor das poções ou de qualquer outra coisa sempre e como seria interessante um dispositivo que pudesse facilitar esse trabalho.

A turma estava bastante agitada - provavelmente por ser a primeira aula no laboratório. Acabamos trabalhando metade do conteúdo previsto. Vale ressaltar, também, que eu poderia ter utilizado melhor a estrutura do laboratório, por exemplo, utilizando o projetor para mostrar os trechos de código.

[5] Um IDE para o Python

[6] Felizmente elas continuavam impossíveis.

Capítulo 7

Aula 6: Variáveis e Tipos

Na aula anterior começamos a desbravar as características do Python. Trabalhamos com impressão na tela e com aritmética. Hoje conheceremos o que são variáveis e tipos.

7.1 Problema da lojinha - versão 1

Um *PokéMart* é uma loja onde se vende alguns itens como *pokébolas* ou mesmo antídotos.

O nosso objetivo é, dada uma quantia de dinheiro, comprar o máximo de pokébolas e, depois, comprar o troco de antídotos. Iremos fazer isso utilizando o Python.

A partir do que foi visto na aula passada, basta fazermos as contas usando os operadores e imprimir o resultado com o `print()`. Para conseguirmos escrever nosso programinha, precisaremos das seguintes informações:

- dinheiro: 3913;
- preço da pokébola: 200;
- preço do antídoto: 100.

Código 7.1: Versão 1

```
1  print(3913) # dinheiro inicial
2  print(200) # preco da pokebola
3  print(100) # preco do antidoto
4
5  # calcular a quantidade de pokebola que podemos comprar: 3913/200 = 19
6  print(3913/200)
7
8  # calcular o troco: 3913 - 200*19 = 113
9  print(3913-200*19)
10
11 # calcular a quantidade de antidotos: 113/100 = 1
12 print(119/100)
13
14 # calcular o troco: 113 - 100 = 13
15 print(119-100*1)
16
17 # imprimindo
18 print("nosso heroi podera comprar 19 pokebolas, 1 antidoto e tera 13 de troco")
```

E se o valor da pokébola mudar para 150?

7.2 Variáveis

Se o valor da pokébola mudar para 150 teremos de refazer **todas** as contas. Além disso, para cada outra mudança, todas as contas terão de ser refeitas **pelos programadores**. Parece bastante trabalhoso trabalhar dessa forma.

Seria bastante legal se existisse algum recurso onde pudéssemos dar um *nome* para os diversos números que aparecem e pudéssemos, também, fazer as contas a partir desses nomes. É mais fácil pensar que a quantidade de pokébolas compradas será dada pelo dinheiro que o herói possui dividido pelo preço da pokébola em vez de pensar que $3913/200 = 19$.

Para nossa alegria, o recurso existe! Chama-se **variável**.

Exemplo:

```
>>> tomate = 100
>>> print(tomate)
100
```

No exemplo acima, criamos uma variável chamada `tomate` e guardamos o número 100.

O grande poder das variáveis é poder trocarmos os diversos números por nomes que conseguimos entender e lembrar.

```
>>> tomate = 100
>>> cebola = 50
>>> vegetais = tomate + cebola
>>> print(vegetais)
150
```

Ainda é possível modificar o valor que está guardado na variável.

```
>>> tomate = 100
>>> tomate = tomate - 3
>>> print(tomate)
97
```

A partir das variáveis podemos transformar as diversas fórmulas que vemos no dia-a-dia em pequenos trechos de código.

```
>>> base = 20
>>> altura = 10
>>> area = base*altura
>>> print(area)
200
```

Resta-nos saber como fazer para os valores das variáveis aparecerem no meio de um `print()`. Na Versão 1 há uma linha que imprime "`nosso heroi podera comprar 19 pokebolas`". Queremos fazer o mesmo, mas utilizando as variáveis.

Podemos utilizar um **marcador** para dizer onde tem uma variável na hora de imprimir. No Python, para imprimir variáveis com **números inteiros**, utilizamos `%d`.

```
>>> gols_do_corinthians = 7
>>> print("O Corinthians fez %d gols."%(gols_do_corinthians))
O Corinthians fez 7 gols.
```

Quando imprimimos utilizando marcadores, precisamos dizer de qual variável queremos o valor. O `print()` passa a conter um trecho extra que diz respeito às variáveis.

Exemplo:

- `print()` normal: `print("19 pokébolas")`
- `print()` com variáveis: `print("%d pokébolas"%(numeroPokebolas))`

Caso desejarmos imprimir mais de uma variável, não há problema. Só precisaremos da quantidade de marcadores adequada.

```
>>> timao = 5
>>> spfc = 0
>>> print("O Corinthians fez %d a %d no São Paulo!"%(timao,spfc))
O Corinthians fez 5 a 0 no São Paulo!
```

É importante ressaltar que a ordem em que as variáveis aparecem dentro do parênteses é a **mesma ordem** em que os marcadores serão substituídos.

7.3 Problema da lojinha - versão 2

Agora que vimos as variáveis, vamos ver como ficaria o problema da lojinha.

Código 7.2: Versão 2

```
1  # versao com variaveis
2  dinheiro = 3913
3  precoPokebola = 200
4  precoAntidoto = 100
5
6  numeroPokebola = dinheiro/precoPokebola
7  print(numeroPokebola)
8
9  dinheiro = dinheiro - precoPokebola*numeroPokebola
10 print(dinheiro)
11
12 numeroAntidoto = dinheiro/precoAntidoto
13 print(numeroAntidoto)
14
15 dinheiro = dinheiro - precoAntidoto*numeroAntidoto
16 print(dinheiro)
17
18 # imprimindo
19 print("nosso heroi podera comprar %d pokebolas, %d antidotos e tera %d ↩
       de troco"%(numeroPokebola,numeroAntidoto,dinheiro))
```

Bem mais fácil de entender! Temos nomes com significado em vez de números feios passeando pelo código.

Em diversos jogos de videogame, o herói possui um nome. Faz um certo sentido pensarmos que asvariáveis armazenam outros tipos de dado como palavras, números decimais etc.

7.4 Tipos de dados

Vimos nas aulas anteriores que somos nós que damos o significado para os zeros e uns do computador a partir de códigos como o ASCII. Da mesma maneira, o Python tem um dispositivo que interpreta determinados uns e zeros como números inteiros, números decimais, letras e outros.

Trabalhamos até agora com números inteiros e números decimais. Veremos como trabalhar com letras e palavras. Chamamos essas palavras de *strings*.

Quando aprendemos a mexer com o `print()`, qualquer coisa diferente de número que mandássemos imprimir tinha de vir entre aspas. Todas essas coisas eram strings!

Podemos guardar strings nas variáveis.

```
>>> heroi = "Naruto"
>>> print(heroi)
Naruto
```

O Python também possibilita algumas operações com strings. Uma delas, a **concatenação**, consiste em juntar duas strings.

```
>>> fruta = "jaca"
>>> fruta2 = "melancia"
>>> print(fruta+fruta2)
jacamelancia
```

As strings também podem conter números, afinal, há uma diferença entre a quantidade **dois** e o símbolo **2** que utilizamos para representar essa quantidade.

```
>>> senha = "h2so4"
>>> print(senha)
h2so4
```

Podemos, inclusive, concatenar strings que só contenham números.

```
>>> dois_mais_dois = "2" + "2"
>>> print(dois_mais_dois)
22
```

Outra operação é a de repetição. Podemos repetir uma mesma string diversas vezes.

```
>>> palavra = "banana"
>>> print("banana"*3)
bananabananabanana
```

Por outro lado, é importante tomar cuidado para não misturar números e strings de números. Caso contrário, obteremos um erro de tipo.

```
>>> print("2"+2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

Dentre o que o Python reconhece como strings estão:

1. Texto entre aspas duplas. Ex: `"Crepúsculo"`;
2. Texto entre aspas simples. Ex: `'Lua Nova'`;
3. Texto entre **três** aspas simples. Ex: `'''Amanhecer'''`.

Como há diversos tipos de dados, faz sentido que tenhamos diversos marcadores. Para imprimir strings usamos o marcador `%s`, para imprimir números decimais, `%f`.

```
>>> heroi = "Vegeta"
>>> print("O nome do heroi eh: %s"%(heroi))
O nome do heroi eh: Vegeta
```

7.5 Relatório

Após a agitação da primeira aula em laboratório, resolvi preparar um material um pouco diferente. Comecei a aula mostrando o exemplo da lojinha, PokéMart, que queria fazer em aula. Depois, mostrei a versão digital do Joguinho de Binário. Tudo isso porque achei que o pessoal não estava tão motivado e, principalmente, porque não soube utilizar os recursos de forma inteligente na aula anterior.

A aula contou com a presença do Vítor que ajudou com a turma.

Vimos o probleminha do PokéMart que é idêntico ao do paladino da aula anterior. O pessoal que estava na semana anterior resolveu quase instantaneamente, mas os alunos que perderam a primeira aula não tiveram o mesmo êxito. A falta de continuidade atrapalha bastante.

Depois, sugeri que mudassem o preço da pokébola para 150 e resolvessem o problema. Em seguida, sugeri que mudassem o dinheiro inicial. Daí começaram as reclamações!

> "Ah! Psor! Mas dá trabalho!"

Mal sabiam eles que era justamente isso que eu queria ouvir. Eu queria ouvir que a atividade era maçante, chata e que seria legal arranjar uma forma esperta de resolver o problema. Começamos a estudar as variáveis.

Rodamos diversos exemplozinhos com variáveis e pedi para que resolvessem o problema do PokéMart utilizando as variáveis. Alguns conseguiram exceto pela impressão, outros não. Posteriormente, vimos como trabalhar a impressão com variáveis.

Aproveitei e sugeri que criassem um nome para o herói, mas um nome a partir de uma variável. Começamos a ver as strings. O ponto alto foi quando mostrei aos alunos que

$$\text{'2'}+\text{'2'} = \text{'22'}.$$

Faltando cerca de 10 minutos para terminar a aula, mostrei aos alunos como abrir o joguinho de binário a partir do IDLE. Deixei-os jogando até o término da aula. Aproveitei para dizer que o jogo tinha um *cheat*. :-D

Não deu tempo de trabalhar com o input() como havia planejado. Contudo, no geral, a aula pareceu produtiva.

Capítulo 8

Aula 7: Entrada de dados

No capítulo anterior, começamos a trabalhar o problema da lojinha. Partimos da solução usual onde o computador atua como uma mera calculadora, depois inserimos as variáveis. O objetivo agora é permitir que o **usuário** modifique os valores.

Em diversas aplicações da vida real como jogos a interação com o usuário é um requisito primordial. Como funcionaria a maioria dos jogos de computador se o jogador não interagisse com o jogo?

Desta forma queremos modificar nossa lojinha para que o usuário forneça as informações de preços e, principalmente, o nome do herói. Todavia não podemos fazê-lo sem antes conhecermos o `input()`.

8.1 *input*

O `input()` é uma função do Python que espera o usuário digitar alguma coisa e guarda em algum lugar. É o mecanismo mais básico de interação com o usuário.

```
>>> idade = input()
99
>>> print("Sua idade eh: %d"%(idade))
Sua idade eh: 99
```

Também podemos deixar uma frase para o usuário antes de pedirmos para ele digitar.

```
>>> titulos = input("Quantos títulos mundiais tem o Corinthians?")
Quantos títulos mundiais tem o Corinthians?65536
>>> print("O Corinthians tem %d títulos."%(titulos))
O Corinthians tem 65536 títulos.
```

Porém, se tentarmos escrever texto, o `input()` falha.

```
>>> nome = input("Qual o seu nome?")
Qual o seu nome?Will
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 1, in <module>
NameError: name 'Will' is not defined
```

Isso acontece porque o `input()` espera esse tipo de texto como se fosse uma string, ou seja, tem de estar entre aspas. Há um jeito mais esperto de obter a entrada sem ter de usar aspas. Podemos usar o `raw_input()`. O comportamento é bem parecido, entretanto, ele diz que tudo o que o usuário digitar será texto.

```
>>> nome = raw_input("Qual o seu nome?")
Qual o seu nome?Will
>>> print(nome)
Will
```

Agora basta trocarmos os trechos adequados no programa da lojinha.

8.2 Problema da lojinha - versão 3

O programador não pode escolher o tanto de pokébolas que o jogador comprará. Precisamos fazer com que esse tipo de coisa venha do usuário e não do programador.

Código 8.1: Versão 3

```
1  # versao com entrada
2  nome = raw_input("Digite o nome do heroi: ")
3  dinheiro = input("Digite a quantidade de dinheiro: ")
4  precoPokebola = input("Digite o preco da Pokebola: ")
5  precoAntidoto = input("Digite o preco do Antidoto: ")
6
7  numeroPokebola = dinheiro/precoPokebola
8  print(numeroPokebola)
9
10 dinheiro = dinheiro - precoPokebola*numeroPokebola
11 print(dinheiro)
12
13 numeroAntidoto = dinheiro/precoAntidoto
14 print(numeroAntidoto)
15
16 dinheiro = dinheiro - precoAntidoto*numeroAntidoto
17 print(dinheiro)
18
19 # imprimindo
20 print("%s podera comprar %d pokebolas, %d antidotos e tera %d de troco"↩
       %(nome,numeroPokebola,numeroAntidoto,dinheiro))
```

A única diferença expressiva da versão 2 para a versão 3 é o `input()` que aparece nas primeiras linhas. Agora as variáveis não recebem um número pré-determinado pelo programador, pois quem as abastece é o usuário.

Também modificamos o programa para trabalhar com o nome do herói.

8.3 Relatório

Definitivamente, a aula que tem tudo para ser excluída nas próximas versões. Pelo próprio material dá para perceber que o progresso foi inócuo.

A ideia original consistia em modificar a versão 2 para trabalharmos com os dados variáveis e, depois, aproveitar o `input()` para fazermos uma versão realmente interativa da lojinha. Entretanto, o que jamais havia me passado pela cabeça é que os alunos já tinham um mecanismo para modificar as variáveis: mexer no código. Isso causou uma tremenda confusão e eu fui incapaz de justificar de maneira convincente o uso do `input()`.

Além disso, o fato de alguns alunos faltarem, a aula ser semanal e o único contato com o ambiente ser em sala de aula contribuem bastante para uma certa deterioração do que foi visto na aula anterior. Tive de gastar um tempo considerável revisando como fazer a impressão.

Aliás, fazer a impressão via marcadores foi uma outra grande besteira. Era mais fácil ter seguido o caminho da concatenação e dos `str()`. Apesar de menos preciso no que concerne à tipagem, é muito mais natural.

Por sorte, contei com a ajuda do André Hirata que aproveitou para conversarmos no pós aula. Principalmente, veio dele a ideia de trabalhar com lista de exercícios simples para verificar de forma mais precisa onde os alunos estão com problemas.

Num nível geral de detalhamento, posso dizer que a aula começou com a revisão da impressão. Gastamos uns 30 minutos. Depois, tentei explicar um pouquinho sobre o `input()`. Uma vez que o conceito foi explicado, pusemos a mão na massa, ou melhor, tentamos.

Há diversas outras formas de se trabalhar interatividade utilizando o Python. Apenas para ilustrar, há a possibilidade de usar algo robusto como o Pygame ou mesmo recorrer às estimadas tartaruguinhas.

Capítulo 9

Aula 8: Implementação da Cifra de César

Nas últimas aulas, tivemos alguma experiência com o laboratório e com o Python. Lembram do que significa "anevm"? Antes de começarmos as aulas no laboratório, vimos um pouco sobre codificações e sobre a Cifra de César. Como será que podemos fazer algo parecido no computador?

Hoje tentaremos entender justamente como escrever um programinha que é capaz de cifrar uma mensagem a partir da Cifra de César.

9.1 Revisão

Vamos lembrar bem por alto como funciona a cifra.

9.1.1 Cifra de César

A Cifra de César é uma estratégia de criptografia cujo funcionamento consiste em rodar para frente as letras do texto em uma quantia pré-determinada de casas. Chamamos essa quantidade de *chave*.

Poderíamos representar essa troca de letras da cifra a partir do dispositivo abaixo. Basta rodarmos o círculo interior e verificarmos as letras correspondentes.



Figura 9.1: Exemplo de diagrama da Cifra de César com $chave = 13$.

O funcionamento da cifra pode ser reduzido a uma simples continha envolvendo letras e

números tal como $A + 1 = B$ ou $A + 5 = F$. Essa operação consiste em *andar* com a letra no alfabeto o número de casas que é *somado*. [1]

Mas os computadores não entendem as letras da mesma forma que nós. O que eles entendem são os zeros e uns. Visitaremos novamente o mundo estranho do **ASCII**.

9.1.2 ASCII

O Código Padrão Americano para Troca de Informações (ASCII) é uma forma de representarmos símbolos como letras e números a partir de *bits*.
Exemplo:

`1100001` representa a letra 'a' no ASCII.

Por outro lado, 1100001 é um número binário e representado em base decimal corresponde ao número 97. Exatamente! Cada letra em ASCII possui uma representação binária e, por conseguinte, uma representação **numérica**. No computador, letras são números!

A tabela abaixo dá uma ideia de qual a correspondência entre as letras em ASCII e os números.

A	B	...	Z	a	b	...	z
65	66	...	90	97	98	...	122

A operação definida acima acaba sendo muito mais natural.
Exemplos:

$$A + 1 = 65 + 1 = 66 = B$$

Isso nos leva à primeira ideia de como fazer a cifra num computador.

9.2 Versão 1

A versão 1 consiste justamente em rodarmos as letras utilizando a representação numérica das letras. Vamos nos ater apenas às letras maiúsculas.

Para transformarmos a letra em sua representação numérica no computador, utilizaremos o `ord()`, por exemplo, `ord('A')` é 65. Para fazermos o inverso, utilizaremos o `chr()`. Se quisermos descobrir qual símbolo é representado pelo número 90, fazemos `chr(90)`, que devolverá Z.

---

para cada letra na palavra
  $numero \leftarrow ord(letra)$
  $numero \leftarrow numero + chave$
  $cifrado \leftarrow chr(numero)$

Embora pareça funcionar, o algoritmo tem um erro curioso. Qual letra vem depois de Z?

Precisaremos dar um jeito de fazer $Z + 1$ virar A. Também precisaremos fazer quaisquer contas que caiam depois de Z caírem nas letras anteriores. Precisamos de um mecanismo parecido com o da Figura 9.1, precisamos de um mecanismo que *dê a volta*.

[1] Se o número for negativo, a operação funciona. Basta andarmos para trás.

9.3 Pizza

Esta seção infelizmente não tem relação com a comida maravilhosa oriunda de Nápoles para homenagear a raínha. Trata-se de uma estratégia de separar números.

Quando dividimos um número, obtemos dois resultados: o quociente e o resto. Geralmente estamos interessados em saber qual o quociente, mas, para montar a pizza, interessaremo-nos pelo **resto**.[2]

Um fato bastante importante acerca do resto de uma divisão é que **ele jamais será maior que o divisor**. Caso contrário, poderia ser dividido novamente e obteríamos um novo resto menor que o divisor.

Vamos ver o resto da divisão **por três** de alguns números.

Número	Resto
0	0
1	1
2	2
3	0
4	1
5	2
6	0
7	1
8	2
9	0

Número	Resto
10	1
11	2
12	0
13	1
14	2
15	0
16	1
17	2
18	0
19	1

Número	Resto
20	2
21	0
22	1
23	2
24	0
25	1
26	2
27	0
28	1
29	2

Podemos dividir esses números numa pizza de três fatias seguindo a regra

"Números que têm o mesmo resto de divisão estão na mesma fatia."

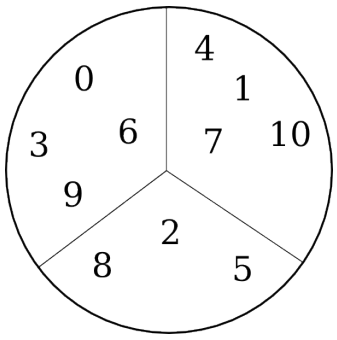


Figura 9.2: Pizza numérica.

De certa forma, dizemos que os números da mesma fatia são equivalentes.

Se trocarmos a pizza de três pedaços por uma pizza de vinte e seis pedaços, onde cada pedaço representa uma letra do alfabeto, resolvemos o problema do Z. Mesmo que o número seja maior que o número correspondente ao Z, basta calcularmos o resto da divisão. O resultado será a fatia correspondente ao número maior que Z.

[2]Urgh! Pizza de resto não!

9.4 Versão 2

Vamos tentar trazer essa estratégia para nosso algoritmo. A ideia é trocar o número pelo resto da divisão caso ele seja maior que o Z.

---

para cada letra na palavra
  $Z \leftarrow ord('Z')$
  $numero \leftarrow ord(letra)$
  $numero \leftarrow numero + chave$
  se $numero > Z$ **então**
    $numero \leftarrow Z\%26$
  $cifrado \leftarrow chr(numero)$

Algo de estranho acontece quando tentamos utilizar o algoritmo acima para cifrar coisas. Símbolos estranhos aparecem! O que aconteceu!?

A explicação para os símbolos estranhos é simples. O resto da divisão é um número **menor** que o divisor, ou seja, um número menor que 26. Como as letras maiúsculas são representadas por números entre 65 e 90, o número que obtemos não corresponde a uma letra.

Se contarmos de 65 a 90 teremos justamente 26 números e mais: é possível associarmos essas letras a números entre 0 e 26 de um jeito bem simples.

letra	representação	número novo
A	65	0
B	66	1
C	67	2
...	...	...
Z	90	25

Agora a "matemágica" acontece! Se fizermos $Z + 9 = 25 + 9 = 34$ e depois calcularmos o resto da divisão por 26, obteremos 8, que corresponde à letra I.

Para desfazer a mudança, basta subtrairmos a chave do número correspondente à nova letra $8 - 9 = -1$ e tomarmos o resto da divisão por 26, que é 25, como queríamos.

Resta agora saber como mudar da representação numérica ASCII para a nossa representação que dá certo. Como `ord('A')` é 65, a conta

$$ord(letra) - 65$$

funciona perfeitamente, pois, se a letra for A, então o resultado será 0. Se fizermos todas as outras contas até Z, observaremos o resultado desejado. Chamamos essa estratégia de **mudança de base**.

9.5 Versão 3

Agora faremos a mudança de base em nosso algoritmo.

---

para cada letra na palavra
  $A \leftarrow ord('A')$
  $Z \leftarrow ord('Z') - A$
  $numero \leftarrow ord(letra)$
  $numero \leftarrow numero - A$
  $numero \leftarrow numero + chave$
  se $numero > Z$ **então**
    $numero \leftarrow Z\%26$
  $numero \leftarrow numero + A$
  $cifrado \leftarrow chr(numero)$

Pronto! Agora temos uma versão funcional da cifra! Por outro lado, diversos detalhes ainda precisam ser considerados.

- Não trabalhamos com letras minúsculas;
- Não trabalhamos com espaços ou outros símbolos diferentes;
- Não cuidamos de como formar o texto traduzido.

Um exemplo um pouco melhor encontra-se disponível em:

`https://linux.ime.usp.br/~gnann/crea/cesar.py`

9.6 Relatório

Antes de escrever qualquer coisa, é importante ressaltar que o conteúdo da aula 8 foi feito em *on the fly*. Houve um problema envolvendo o laboratório e acabei sendo avisado completamente em cima da hora.

Comecei a aula perguntando se alguém lembrava o que significa "anevm". Um aluno que estava presente na aula 4 disse que sim. Disse que significava nariz. Outros alunos não acreditaram muito, porque não vieram à aula 4, mas logo disse-lhes que sim, que "anevm" era nariz.

Contei que o objetivo a aula era estudar como escrever essa cifra num computador. Relembramos rapidinho como funcionava a Cifra de César e, depois, revimos o ASCII. Por fim, vimos que é possível somar inteiros em letras ASCII.

Começamos a tentar conceber o algoritmo. Um aluno disse que a primeira coisa a fazer era pegarmos o `input` do usuário. Outro disse que deveríamos trabalhar letra por letra e utilizar a soma que acabamos de ver.

Trouxe o problema do Z, mas ninguém teve uma ideia para resolver. Expliquei-lhes sobre as classes de equivalência a partir da pizza. Para tanto, começamos fazendo umas divisões na lousa. Depois argumentei que, de fato, não poderíamos ter um divisor menor que o resto.

Vimos assim que os números poderiam ser guardados nas fatias da pizza a partir do resto da divisão. Concluímos, ainda, que poderíamos colocar qualquer número inteiro numa das fatias.

Pedi para um aluno desenhar uma pizza de 26 pedaços na lousa. Perguntei a ele se entendera o objetivo. Ele disse que parecia ser algo como usar o fato de todo número estar numa fatia da pizza para encontrar a letra que Z deveria virar. Entretanto, não soube explicar o porquê.

Tentamos novamente resolver o problema, mas não obtivemos sucesso. Faltava, ainda, mudar a base. Escrevi a bijeção entre as letras e os números de 0 a 25. Nessa altura, alguns alunos já dormiam e outros queriam ir embora. Aulas não práticas pareceram bem menos divertidas para eles, não obstante um dos alunos ter dito a mim justamente o contrário no começo da aula.

Terminamos o algoritmo, agora a ideia era verificar se funciona. Pedi que uns alunos fossem à lousa testar se a letra era rodada direitinho. Não tivemos problemas para simular o algoritmo.

Fortuitamente, a aula terminou. O pesadelo acabara.

Se fosse eleger alguma aula para ser a pior do semestre, essa possivelmente ganharia o prêmio. Apesar de reconhecer a importância, certamente seria mais prazeroso trabalhar parte das coisas em laboratório. Suspeito, inclusive, que os alunos também concordariam. Sem contar que poderiam ver o programinha funcionando e até enviar mensagens cifradas para seus colegas.

Capítulo 10

Aula 9: Corrida de Tartaruga!

Um dos grandes atrativos para se trabalhar com Python é justamente a biblioteca Turtle. Em linhas gerais, é um recurso lúdico para programar. O objetivo é movimentar a tartaruguinha e desenhar alguma coisa (mas também podemos apostar corrida!).

10.1 Atividade: Corrida de Tartaruga!

10.1.1 Objetivos

Esta atividade tem como objetivos:

- Trabalhar com a biblioteca Turtle;
- Instigar o espírito competitivo dos alunos.

10.1.2 Descrição

Há um circuito num formato quase oval e precisamos fazer a tartaruga dar uma volta nesse circuito. Ganha quem der a volta mais rápida!

10.1.3 Regras

O campeonato possui as seguintes regras:

- Ganha quem der a volta mais rápida;
- Se sair da pista, está desclassificado;
- A distância máxima que pode se movimentar por comando é 50.

10.1.4 Instruções

Para abrir o circuito, precisaremos abrir um terminal e digitar `python`. Lá basta digitar

```
from esqueleto import *.
```

Para movimentar a tartaruga temos os comandos abaixo.

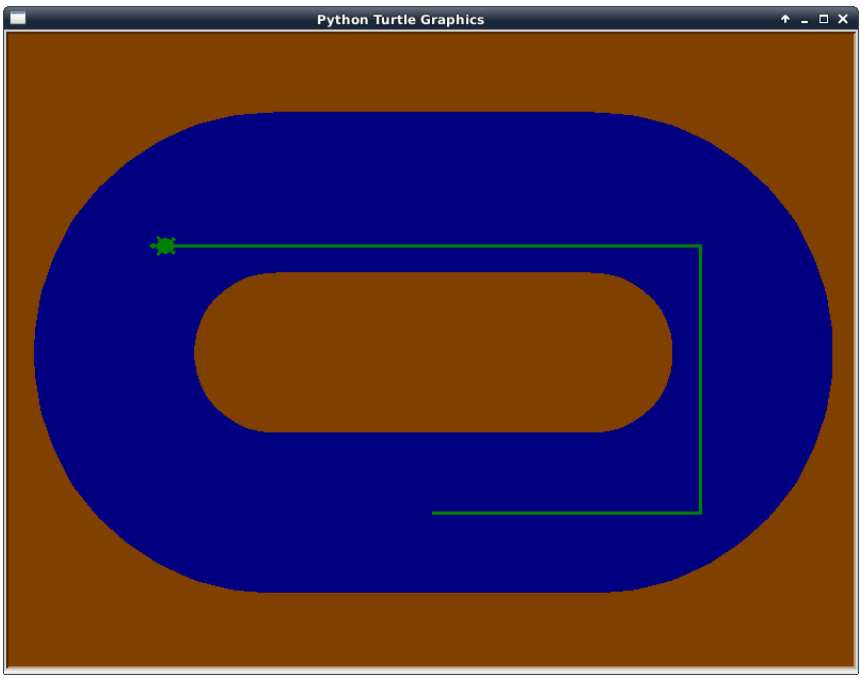


Figura 10.1: Circuito do campeonado.

Comando	Comportamento
`t.forward(n)`	anda n para frente
`t.back(n)`	anda n para trás
`t.left(n)`	gira n para a esquerda
`t.right(n)`	gira n para a direita
`t.reset()`	volta para o começo

Uma vez terminada a volta, salve os comandos num arquivo e chame o professor.

10.2 Relatório

Ah! A corrida de tartaruga! Uma atividade voltada para o evento do Dia das Crianças que tinha tudo para dar certo. Até deu um pouco certo.

Escrevi a tabelinha com os comandos na lousa, depois mostrei como fazer para importar o programinha com o circuito.

A atividade em si foi muito boa. Todos alunos participaram, alguns até disseram que um jogo que jogam, *Minecraft*, possui um recurso parecido com a tartaruga.[1]

O fato mais curioso é que alguns alunos ignoraram o campeonato e ficaram fazendo a tartaruga passear por aí...

Os problemas foram em sua maioria de ordem tecnologica.

10.2.1 Editor de texto

Esqueci-me de verificar se a distribuição que eu tinha montado tinha um editor de texto decente para modo texto. Muitos dirão que `gvim` e `emacs` são ótimos. Não questiono, mas seria legal uma coisa simples para o público alvo.

Acabou que tivemos de usar o Libre Office Writer para meramente copiar uma lista de comandos do tipo `t.forward(50)`. Devia ter tomado mais cuidado e especificado certinho **como** os alunos fariam para me passar as coisas.

10.2.2 $14 * 2 = 28$

Colocar um pen drive num computador e pegar um arquivo é uma tarefa que despende $O(1)$ de tempo se o pen drive funciona. Colocar um pen drive em n computadores é uma tarefa que despende $O(n)$ de tempo. Numa aula de 80 minutos, não é razoável gastar uma parcela significativa para salvar os arquivos dessa forma.

Se tudo der certo, vou ver se escrevo algum sisteminha bobo de entrega de atividades. Mas tenho de tomar cuidado para não liberar o monstro da Internet. Não que as coisas do curso sejam chatas, mas é *bem* difícil competir com a interatividade e babaquice de coisas como as redes sociais e os canais de vídeos.

Fui praticamente o último a sair da escola porque fiquei pegando arquivos nos computadores. O campeonato não pode ser avaliado na frente dos alunos. Pior: não pude pegar todos os circuitos dos alunos presentes. Cancelei o campeonato e premiei a todos os participantes.

10.2.3 Tk

O problema de usar um IDE baseado em Tk e escrever aplicações baseadas em Tk é bem simples: se um dos dois travar o Tk, o Tk cai para ambos. Isso não parece ser razoável.

Para uma próxima versão preciso muito correr atrás de alternativas para ambiente gráfico que não chorem na primeira exceção que encontrarem.

Como é terrível ver que, apesar de ter montado o circuito e verificado de certa forma a usabilidade, acabei negligenciando diversos detalhes como os acima. Ademais, a quantidade de *bugs* foi alta. Dividir-me entre as mais de dez pessoas presentes não foi tarefa fácil.

Fico pensando se não acabei estragando o Dia das Crianças...

[1] E possui mesmo!

Capítulo 11

Aula 10: Exercícios

11.1 *print*

1. Escreva no **interpretador** o seguinte comando: `print("ALO ALO ALO DAORA")`.
2. Escreva no interpretador um *print* que imprima a seguinte frase:
   `SPFC: 27 pontos em 25 jogos`.
3. Escreva no interpretador um *print* que imprima o que você quiser.
4. Escreva no interpretador um *print* que imprima a seguinte frase:
   `Joseph disse: "ALO ALO ALO DAORA"`.
5. Escreva no interpretador um *print* que tenha duas linhas.

11.2 Operadores

1. Escreva no interpretador $1 + 2 * 3$.
2. Calcule, com a ajuda do interpretador, a seguinte conta: $2 + 2$.
3. Calcule, com a ajuda do interpretador, a seguinte conta: 2^{100}.
4. Calcule no interpretador as seguintes contas: $1/2$ e $1.0/2$. Qual a diferença entre elas?
5. Quanto é "dois mais dois sobre três?"
6. Calcule no interpretador a seguinte conta: $200/0$.
7. Quanto é o resto da divisão de 5 por 2?
8. Escreva no interpretador 5%2. Será que % calcula o resto da divisão?

11.3 Variáveis

1. Escreva no interpretador `var1=77`. Depois, escreva `print(var1)`.
2. O que aconteceu na linha acima?
3. O que é uma variável?

4. Escreva no interpretador `pokemon="Charizard"`.
   O que acontece se escrevermos `pokemon=Charizard`?

5. Escreva no interpretador:
```
a = 10
b = 20
c = a+b
print(c)
```

6. Escreva no interpretador:
```
a = 10
print(a)
a = 12
print(a)
```

7. Escreva no interpretador:
```
cachorro="Snoopy"
print(Cachorro)
```

8. Escreva um **programinha** que tenha uma variável com o valor "Charizard" e imprima essa variável.

9. Escreva no interpretador:
```
pokemon = "Charizard"
print(pokemon)
print("pokemon")
```
   Qual a diferença entre os *prints*?

10. Escreva um programinha que uma variável chamada `base` valendo inicialmente 10 e outra chamada `altura` valendo inicialmente 20 e imprima o **produto**[1] delas.

11. Modifique seu programinha anterior para calcular a área de um **quadrado**.

12. Escreva no interpretador:
```
a = 10
print("Tenho %d tartarugas."%(a))
```

13. Escreva no interpretador:
```
alfafa = 10
print("Tenho %d tartarugas."%(tartarugas))
```
   Por que deu errado?

14. Escreva no interpretador:
```
VARIAVEL = 112
print("Tenho %d cebolas."%(variavel))
```
   Por que deu errado?

15. Escreva no interpretador:
```
heroi = "Naruto"
print("Meu heroi chama %s. Não é daora?"%(heroi))
```

16. Escreva um programinha contendo as seguintes instruções:

[1] Lembre-se! Produto é multiplicação!

```
nomePokemon = "Mewtwo"
levelPokemon = 100
print("Meu pokemon chama %s. Lv: %d"%(nomePokemon,levelPokemon))
```

17. Escolha um pokémon e um *level* e troque no seu programinha anterior.

18. Escreva um programinha com três variáveis:

    - `nome="Cesar"`
    - `idade=56`
    - `cidade="Roma"`

    O programinha deve imprimir a frase:

    "Nome: <nome>. Idade: <idade>. Cidade: <cidade>."

    Onde <nome> é o conteúdo da variável **nome**, <idade> é o conteúdo da variável **idade** e por aí vai...

11.4 Relatório

Depois do descalabro das aulas anteriores, fiquei pensando como teria sido melhor se eu pudesse ter dado a aula de exercícios quando a programei. Era para ela ter sido a aula 8. De fato, teria sido melhor.

Distribuí as listas de exercício para os alunos e perguntei se eles queriam que eu começasse a ler em conjunto. Assentiram e comecei a ler o enunciado da questão trivial. As não triviais eu não li. `:-)`

Dado o número reduzido de alunos deu para atender todos sem muitos problemas. Inclusive nem houve tantos problemas. A causa mais comum para atendimento era algum problema de sintaxe que não perceberam.

Alguns exercícios que se mostraram interessantes durante a aula foram:

- 1.5: não havia dito a eles como imprimir em mais de uma linha com apenas uma string;
- 2.3: alguns alunos gostaram do número;
- 2.5: ambos resultados possíveis apareceram;
- 3.4: alguns não perceberam que não tinha aspas;
- 3.14: problemas com a diferença de caixa;
- 3.18: a sintaxe ¡nome¿ não foi compreendida.

Cabe ressaltar que eu tinha uma lista B caso algum aluno terminasse todos os exercícios durante a aula. Um deles terminou e tentou fazer a fórmula de Bháskara no Python.

Tive mais sorte que nas aulas anteriores. As coisas correram razoavelmente bem.

Capítulo 12

Aula 11: Condicionais

Por diversas vezes já nos deparamos com situações que envolviam condições. Quem nunca foi ameaçado em ficar de castigo pela eternidade **se** não passasse de ano?

Se o ônibus passar logo, eu não me atrasarei.

Se você for uma boa criança, Papai Noel te dará presente.

Se o Corinthians ganhar a Libertadores, o Palmeiras cai!

Os condicionais estão por toda parte. Eles estão até em linguagens de programação! Também estarão no tema da aula de hoje.

12.1 *if*

Como os condicionais da vida real, os condicionais da programação submetem trechos do código a alguma condição prévia. O `if`, **se** em Português, é a estrutura condicional em Python. Exemplo:

```
>>> if (2 < 3):
...     print("Sucesso!")
...
Sucesso
```

Note que o `print("Sucesso!")` do exemplo acima ficou um pouco à direita do `if`. Chamamos essa diferença de margem de **indentação**.

A indentação serve para explicar para o Python que o trecho de código indentado está subordinado ao `if`, ou seja, só executará se a condição do `if` for verdadeira.

```
>>> if (1 > 0):
...     print("Sucesso!")
...
>>>
```

Além disso, é importante ressaltar que a quantidade de espaços para indentar um trecho tem sempre de ser a mesma, senão obteremos um erro de indentação.

```
>>> if (0 < 1):
...   print("indentação")
...     print("errada")
  File "<stdin>", line 3
    print("errada")
    ^
IndentationError: unexpected indent
```

O objetivo do erro é garantir que o que está indentado está subordinado a um mesmo condicional.

```
>>> if (0 < 1):
...     if (1 < 2):
...         print("dentro de dois ifs!")
...     print("dentro de um if!")
...
dentro de dois ifs!
dentro e um if!
```

No exemplo acima, se o Python não fosse exigente quanto à indentação, não conseguiríamos distinguir o que está subordinado ao primeiro `if` do que está subordinado ao segundo `if`.

12.2 Condições

O que fica entre parênteses depois do `if` é a **condição**. No primeiro exemplo, verificamos se 2 é menor que 3, no segundo, se 1 é maior que 0. Caso simplesmente digitemos algo assim no interpretador, o Python nos devolverá **True** (verdadeiro) ou **False** (falso) dependendo da sentença.

```
>>> 2 < 3
True
>>> 3 < 2
False
```

Além de desigualdades, podemos verificar, também, se dois valores são iguais ou diferentes. Utilizamos `==` para verificar a igualdade e `!=` para a diferença.

```
>>> "cenoura" == "batata"
False
>>> "cenoura" == "cenoura"
True
>>> "batata" != "cenoura"
True
>>> "batata" != "batata"
False
```

Justamente porque podemos usar livremente o comparador de igualdade - seja numa condição de um `if` ou mesmo para verificar se uma sentença é verdadeira ou falsa - **não utilizamos** o símbolo `=` como comparador. O símbolo `=` é o responsável por **atribuirmos** valores às variáveis.

```
>>> x = 3
>>> print(x)
3
>>> x == 3
True
```

No exemplo acima, primeiro dizemos que a variável x vale 3, depois comparamos o valor da variável x com 3 e o Python nos devolve verdadeiro, pois x vale 3.

De forma resumida, a tabela abaixo contém os principais comparadores utilizados em Python.

operador	descrição
==	igual
!=	diferente
>	maior
<	menor
>=	maior ou igual
<=	menor ou igual

12.3 Lógica (de novo!)

Para que ninguém sinta saudades, está de volta a Lógica. Como os comparadores trabalham com verdadeiros e falsos é natural que os conectivos da Lógica exerçam seu papel nos condicionais.

A partir deles, podemos construir condicionais onde diversas condições precisam ser satisfeitas. Por exemplo, na frase

Se o ônibus passar logo e *eu estiver no ponto, não me atrasarei.*

duas condições precisam ser satisfeitas:

1. O ônibus precisa passar logo;
2. Eu preciso estar no ponto.

Esse e é o mesmo que vimos na aula sobre Lógica! Para usá-lo em Python e juntar duas ou mais condições, utilizamos o **and**.

```
>>> if (0 < 2 and 3 == 3):
...     print("Que exemplo horroroso!")
Que exemplo horroroso!
```

Assim como temos o e, também teremos o ou que, em Python, é representado por **or**.

```
>>> if (0 < 2 or 3 != 3):
...     print("Esse é mais horroroso ainda.")
Esse é mais horroroso ainda.
```

É importante perceber que a condição `3 != 3` não é verdadeira. Contudo, como o ou requer que pelo menos uma seja satisfeita, o resultado é impresso.

12.4 *if*, *else* e *elif*

Na saga pelo domínio dos condicionais ainda resta o `else`. O que estiver subordinado ao `else` executará **somente** se a condição do `if` for falsa. Em Inglês, `else` significa **senão**.

```
>>> if (2 < 0):
...     print("Dois é menor que zero!")
... else:
...     print("Não! Zero é menor que dois!")
...
Não! Zero é menor que dois!
```

Para terminar a aventura condicional, veremos o `elif`, que significa **else if** (senão se). Ilustraremos a diferença nada sutil entre utilizar diversos `ifs` e utilizar o `elif`.

Código 12.1: Vários ifs

```
1 x = 3
2 if (x < 2):
3     print("menor que dois!")
4 if (x < 4):
5     print("menor que quatro!")
6 if (x < 6):
7     print("menor que seis!")
```

No exemplo acima, teremos a seguinte saída:

```
menor que quatro!
menor que seis!
```

Código 12.2: elif

```
1 x = 3
2 if (x < 2):
3     print("menor que dois!")
4 elif (x < 4):
5     print("menor que quatro!")
6 elif (x < 6):
7     print("menor que seis!")
```

Já no exemplo acima, a saída será:

```
menor que quatro!
```

O Python verificará se $x < 4$ **somente** quando a condição $x < 2$ for falsa.

12.5 Relatório

A aula 11 foi realizada no Dia dos Mortos. Pouca gente compareceu.

Diferentemente do material de aula, resolvi mesclar trechos da aula com trechos da aula posterior. Acabei começando a aula perguntando se o pessoal sabia jogar Jogo da Velha. Todos disseram que sim, apesar de alguns dizerem que são bons.

Convidei-os para umas partidas na lousa para observarmos o jogo e, depois de umas cinco partidas, perguntei quais são as regras. Todas as respostas envolviam o próximo objeto de estudo e eram parecidas com a abaixo.

> ***Se*** *formar uma linha na vertical, na horizontal ou na diagonal somente com X ou O, o jogador vence.*

Como todos sabiam que o objetivo era montar o Jogo da Velha, disse-lhes que em Python existe um recurso parecido com esse **se**.

Experimentamos um pouco os `ifs` e aconteceu uma chuva de erros de sintaxe. Os alunos, mesmo eu tendo explicado, confundiam-se com o `=` e o `==`. O que é bastante natural (eu mesmo vivo errando). Parei a experimentação e discutimos a diferença entre os iguais.

Depois, criando uma variável para cada casa do tabuleiro, tentamos simular uma situação de vitória no Jogo da Velha envolvendo os condicionais.

Como sobraram 10 minutos, deu tempo de eu mostrar que o Jogo da Velha sempre acaba empatado se o jogador souber o que está fazendo.

Nota importante: quanto mais cedo puder preparar o laboratório, melhor!

Capítulo 13

Aula 12: Jogo da Velha

Depois de um árduo caminho pelo submundo dos computadores, chegamos ao último chefão, o Jogo da Velha. O objetivo é tentar, dado o tabuleiro graficamente, modelar e construir um Jogo da Velha digital.

13.1 Informações gerais

O Jogo da Velha é um passatempo muito popular no Brasil com regras bem simples, onde dois jogadores marcam alternadamente um tabuleiro 3x3 com símbolos diferentes. Ganha aquele que conseguir formar uma linha vertical, horizontal ou diagonal com três símbolos iguais.

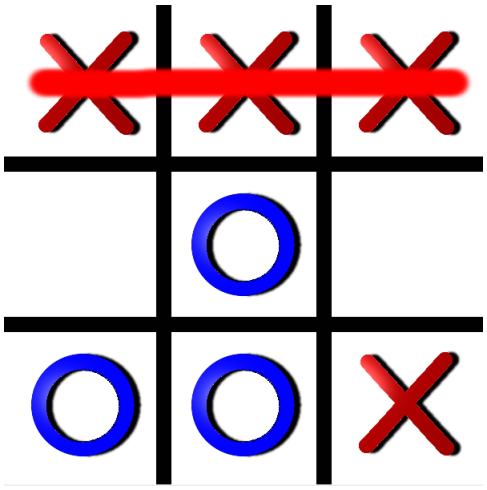


Figura 13.1: Jogador X vence!

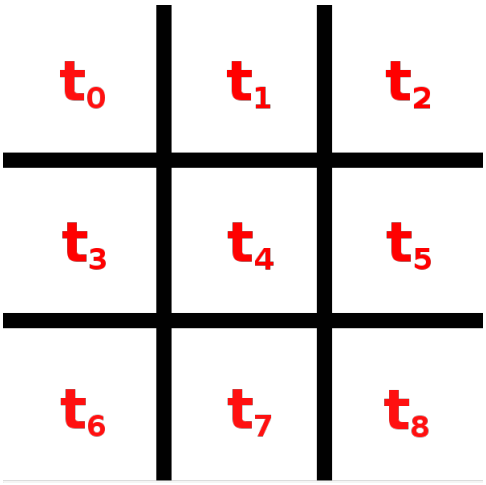


Figura 13.2: Posições do tabuleiro.

13.2 Modelo

O modelo que utilizaremos para tabuleiro nesta atividade é bem simples. Numeraremos as posições de t_0 a t_8 como acima. Para cada posição, teremos uma variável que armazenará a jogada.

Já para os jogadores, precisaremos de uma variável que controla qual o jogador atual e, por conseguinte, auxilia na mudança de turnos.

Por compatibilidade, recomenda-se a criação de uma variável chamada `fim=False` e outra chamada `velha=False`. Ambas são utilizadas pelo parte gráfica.

13.3 Jogo

O arquivo `velha.py` possui dois trechos com comentários dizendo onde devemos salvar as variáveis e a partir de onde poderemos escrever a lógica.

Para escrevermos o jogo, precisaremos resolver, em princípio, três problemas:

1. Como colocar as jogadas no tabuleiro?
2. Como trocar o turno do jogador?
3. Como saber se o jogo acabou?

Além disso, foi fornecida uma variável, `posicao`, cujo conteúdo após um clique é exatamente qual a posição do tabuleiro foi clicada. Sugere-se que o valor armazenado na posicao para escolher em qual casa será feita a jogada.

13.3.1 Turno

Para criarmos o turno, precisaremos pegar o resultado do clique que já está na variável `posicao`, escolher a variável de tabuleiro adequada e marcar com X ou O do jogador atual.

Feita a jogada, é preciso saber uma de duas coisas:

- O jogo acabou?
- Qual o próximo jogador?

13.3.2 Vitória

Um jogador vence o jogo no momento em que há uma linha vertical, horizontal ou diagonal preenchida por X, se o jogador for o X, e por O caso contrário. Basta verificarmos essas **condições** e, caso necessário, atualizar a variável `fim` para **True**.

Dica:

```
if (t0 == "X" and t1 == "X" and t2 == "X"):
    fim = True
```

13.3.3 Velha

Por outro lado, é possível nenhum jogador vencer. Isso acontece quando todas as posições do tabuleiro estão preenchidas e nenhuma linha foi formada. Precisamos cuidar disso. Ademais, recomenda-se atualizar a variável `velha` para **True** para aparecer o resultado correto.

13.3.4 Troca

Depois de um clique, só podemos trocar de jogador se a jogada feita for válida. Para tanto, é necessário que, ao fazer a jogada, o jogador tenha escolhido uma posição **vazia**.

13.4 Relatório

Na realidade, a aula 12 se dividiu em duas partes:

- Cuidar do tabuleiro e dos turnos;
- Cuidar do fim de jogo e aperfeiçoamentos.

Comecei a aula revisando as regras do Jogo da Velha e mostrando aos alunos uma versão funcional do projetinho que iríamos desenvolver.

Depois, vimos como seria o modelo do tabuleiro utilizando oito variáveis. Todo mundo pareceu entender, inclusive aqueles que não estavam na aula anterior.

Pedi para que criassem as variáveis relativas ao tabuleiro e dei um exemplozinho de como deveriam ficar. Nenhum problema para criarmos as variáveis! Com sorte, o pessoal deve ter entendido mais ou menos para que elas servem.

Expliquei qual era a ideia. Primeiramente colocaríamos X nas variáveis só para ver como o tabuleiro se comportaria. Deixei-o pronto para, assim que mudar o valor da variável, colocar a figura relativa à jogada na tela.

Colocaram os Xs no tabuleiro. Agora precisávamos fazer rodar o turno. Alguns tentaram trocar X por O, outros colocaram O nas casas pares... Teve de tudo! Exceto quem acertou de primeira.

Aproveitei o ensejo para explicar qual era o mecanismo de troca de turno. Consistia em criar uma variável dizendo quem é o jogador atual e trocar. Como ainda não havia falado do `elif`, o pessoal penou um pouquinho porque acontecia isto:

Código 13.1: elif

```
1 if (jogadorAtual == "X"):
2     jogadorAtual == "O"
3 if (jogadorAtual == "O"):
4     jogadorAtual == "X"
```

Mostrei o `elif` mesa a mesa e o pessoal começou a ter um jogo que trocava de turnos! Porém o jogo ainda não termina. Fim da primeira parte.

O ruim de dar aulas *single-player* é o fato de não ser possível atender uma quantidade grande de alunos apanhando da sintaxe. Ainda é um problema trocar 'x' por 'X', escrever PRINT... Por outro lado, o pessoal se aventurava mais em mexer numas variáveis e acrescentar umas coisas.

Comecei a segunda parte revisando o que havíamos feito, pedindo encarecidamente para que o pessoal mantivesse o lugar escolhido na aula passada e listando o que precisávamos fazer para a janelinha de cliques com X e O virar um Jogo da Velha.

Alguns alunos já haviam me falado na semana passada que o jogo não terminava, então a primeira coisa que fizeram foi me lembrar disso.

Redesenhei o tabuleiro na lousa, reescrevi as posições e começamos a discutir como fazer para ver se o jogo terminou. Mostrei a sintaxe do `if` com **and** e, logo depois, alguns alunos conseguiam ganhar e perder jogos. Somente ganhar e perder. Nada de empatar.

Um aluno conseguiu terminar uma versão ok do jogo sem mais nenhum empurrãozinho. Isso foi bonito. Daí ele pediu para que eu testasse o jogo, mas tinha um *bug* simples: ainda não havíamos escrito o mecanismo de coibir jogadas em casas iguais, logo era possível sobrescrever jogadas! :-D

Outro problema que apareceu com o pessoal foi o trocar o turno depois de processar a última jogada. Daí o vencedor sempre era o vencido. Trocamos uns `ifs` de lugar, acrescentamos uns `elifs` e tudo começou a funcionar.

Outro aluno reclamou que o esquema com `if` era muito chato e demora demais.

Para minha surpresa, o pessoal conseguiu terminar relativamente cedo o trecho de determinar quem venceu. Daí tentaram regularizar as jogadas.

Deu 13h00, quase todo mundo tinha uma versão jogável. Com bugs! Mas jogável.

O balanço geral do Jogo da Velha indica que coisas parecidas parecem funcionar bem. Por outro lado, tive poucos alunos no segundo dia. Chuva, fim de semestre, prova da ETEC no dia 30...

Capítulo 14

Conclusão

Ao realizar este trabalho, diversos resultados interessantes foram obtidos e, principalmente, a experiência de tê-lo feito talvez seja seu maior legado.

Montar e aplicar um curso de computação é uma tarefa terrivelmente mais difícil do que parece. Primeiramente porque a referência é escassa. Não há uma panaceia para montar o curso perfeito. Ademais, existe uma diferença abissal entre o comportamento da turma na mente de quem está escrevendo um livro e o comportamento da turma numa aula de verdade. Essas características têm de ser levadas em consideração. Por fim, a maioria da literatura trata do uso de computadores no ensino e não do ensino de computação em si.

Por outro lado, pareceu bem claro que, com as devidas correções, é possível dar um curso de computação divertido e proveitoso para o público do ensino fundamental. Entretanto, parece imprescindível um meio de não deteriorar o conteúdo visto entre as aulas. O fato de termos aulas somente aos sábados atrapalhou bastante a compreensão de alguns conceitos. Os alunos, em geral, não tinham contato com o conteúdo fora da aula de computação, logo acabariam esquecendo de detalhes.

Por se tratar de um curso que anda perto da tecnologia, fazer bom uso dela se faz completamente necessário. O uso de Linux nos computadores criou um ambiente diferente do comum e contribuiu indiretamente com o andamento do curso. Porém, detalhes que foram desconsiderados no planejamento das aulas interferiram negativamente. Por exemplo, tanto o IDLE quanto o Turtle trabalham com o TkInter, um acabou atrapalhando o outro quando trabalhamos com o Turtle.

Diversos conteúdos das aulas que foram dadas durante este projeto podem ser recombinados a fim de aproveitar melhor o tempo e os recursos.

Por fim, a partir de uma pesquisa informal feita em sala de aula, pude concluir que, aos olhos dos alunos, ter feito o curso foi melhor do que não ter feito nada. Um alento para a confecção das próximas versões!

Capítulo 15

Parte subjetiva

15.1 Desafios e Frustrações

Fazer um trabalho de conclusão de curso é certamente uma coisa que traz consigo diversos desafios.

O primeiro deles é lidar com um prazo engessado tal como o da disciplina. Por um lado, os alunos são desencorajados a terminar o TCC num semestre, por outro, também são desencorajados a pegar projetos grandes, afinal, não há tempo o suficiente. Em vez de dizer que o TCC é o trabalho que mostra o aluno em seu esplendor, posso dizer que a impressão que fica é a de um obstáculo que foi superado.

Outro problema, pelo menos para mim, são as datas sugeridas. Apesar de elas serem razoavelmente confortáveis, acabam atrapalhando. Talvez seja melhor ser um pouco mais *caxias* com as datas para apresentar coisas como orientador, projeto, prévia etc.

Agora, deixando o abstrato de lado, vamos falar sobre o TCC em vez de falarmos sobre TCC.

A principal motivação que me levou a tentar esse tipo de TCC é o fato de uma das coisas mais agradáveis de trabalhar ter sido dando aula de operação de computadores no EJA. Além disso, o TCC parecia uma coisa mais aberta a esse tipo de coisa social. Não sei o quanto é razoável colocar essas tarefas conhecidas por *extensão* dentro de dissertações e teses.

Um desafio grande para montar meu trabalho foi ter uma perspectiva de que o curso que eu montaria teria uma alma. Quando surgiu o esquema do Crea+, pensei em, pelo menos, trazer diversos conceitos que apareciam no material sem computadores. Teria como verificar se as coisas funcionariam de fato. Teria como ver o que faz sentido e o que não faz.

Outro desafio foi formatar as coisas para que as aulas fossem minimamente agradáveis ao público. Diversas vezes foi nítido que o pessoal não estava tão interessado, entretanto, sempre teve uma parcela da turma que aproveitava mesmo os momentos mais maçantes.

Devo atentar que o mais difícil de todos os desafios é de fato sentar para fazer o trabalho. É bem difícil equacionar o tempo entre trabalho, vida, outras coisas da graduação e o TCC. Inclusive, reprovei a versão anterior por isso. Também desisti da versão anterior por deixar as coisas para depois e perceber que o resultado ficaria bem aquém do que eu gostaria.

As frustrações são mais de caráter pessoal. Parte delas já foi descrita nos relatórios de aula, mas, o que vale ressaltar é a sensação de ter estragado tudo e de poder ter feito algo melhor, escolhido as coisas de uma maneira melhor ou ter atentado aos detalhes.

Também fica uma certa frustração por eu não ter chegado a um formato definitivo para o conteúdo dos pen drives. Com sorte, nas próximas versões, tenho um modelo mais estável. Se bem que o sistema de entrega de tarefas parece bastante importante depois da aula das tartarugas.

15.2 Disciplinas

Estendendo o que foi pedido, falarei, inclusive, das disciplinas irrelevantes.

15.2.1 Relevantes para o TCC

Algumas disciplinas relevantes do BCC para o trabalho.

MAC0110 - Introdução à Computação

Por se tratar de uma proposta de curso de computação, nada mais razoável do que levarmos bastante em consideração o que foi visto e principalmente a forma do curso de MAC0110.

MAT0138 - Álgebra I para Computação

Uma disciplina que envolve diversos conceitos imporantes como classes de equivalência e números inteiros. Apareceu indiretamente em diversas aulas.

MAC0335 - Leitura Dramática

Como a proposta do curso é montar e aplicar um curso, uma disciplina como esta teve muita importância. Saber falar e, principalmente, falar com o público é fundamental.

MAC0329 - Álgebra Booleana e Aplicações

Uma disciplina que trabalha os uns e zeros tem papel importante na hora de explicar como funcionam os circuitos que deram origem aos computadores.

MAC0414 - Linguagens Formais e Autômatos

A disciplina que trabalha a base da computação, ao meu ver. Ter contato com autômatos (e máquinas de estado em geral) parece imprescindível a qualquer estudante de computação justamente porque a base de toda teoria da computação reside nessas máquinas. Usei na aula 1 para ilustrar a computação teórica.

15.2.2 Relevantes

Aproveitando o ensejo, vou aproveitar para deixar uma lista de disciplinas que foram importantes para mim.

MAT0139 - Álgebra Linear

Provavelmente foi o primeiro grande desafio do curso. Aqueles espaços afins do inferno deram o que falar, inclusive deram mais 70% de reprovação. Afora o trauma, é inegável dizer que a Álgebra Linear está praticamente em todo lugar.

MAC0122 - Princípios de Desenvolvimento de Algoritmos

Todo mundo sabe que 122 é fundamental e, de fato, não dá para pensar em ter feito o curso sem 122.

MAT0213 - Álgebra II

A disciplina mais divertida da graduação. Desenvolve um jeito de pensar que nenhuma outra disciplina do curso fez com primazia.

MAC0300 - Métodos Numéricos da Álgebra Linear

Traz aplicações bonitas de AlgeLin. Também trabalha com algoritmos do ponto de vista numérico. Particularmente, ensinou a resolver um problema que eu sempre quis resolver: solução de sistemas lineares de tamanho arbitrário. Acabei, inclusive, fazendo Análise Numérica por causa de 300.

MAC0458 - Direito e Software Livre

É uma disciplina importante para a vida. Todo mundo devia ter alguma noção de direito. Essa disciplina conseguiu cumprir esse requisito incluindo ainda aspectos sociais.

15.2.3 Irrelevantes

Por outro lado, parece igualmente interessante falar sobre disciplinas que nada ajudaram no curso.

4310126 - Física I

Talvez sirva para formação geral, entretanto não vejo diferença entre estudar Física ou qualquer outra área. Parece-me um absurdo que diante de tanta coisa importante - como redes de computadores - uma disciplina como esta seja obrigatória.

4310137 - Física II

Se Física 1 servia para formação geral, não sei se posso dizer o mesmo sobre Física II. É um enorme desperdício de tempo e ainda impede as pessoas de buscarem uma formação mais diversificada por se tratar de uma disciplina obrigatória. Apesar de compreender que, na criação do curso, era uma disciplina razoavelmente fundamental, também reconheço que desde a década de 1970 o mundo mudou. Não faz sentido ter um fantasma do passado como esses no curso.

MAT0211 - Cálculo Diferencial e Integral III

Essa disciplina sequer devia ser do departamento de Matemática Pura. Ela deveria ser chamada de algoritmos para resolução de integrais múltiplas em duas ou três dimensões. Não ajudou em nada. Não introduziu uma maneira de pensar. Só serviu para gastar tempo.

FLC0474 - Língua Portuguesa

Ainda não entendi qual o propósito da disciplina. Uns dizem que é para a garotada aprender a escrever, outros dizem que é para os alunos do BCC terem contato com algo que seja fora de exatas. Não serviu para ambas as coisas.

15.3 Continuação

Pretendo continuar no Crea+. A primeira coisa a fazer é levantar os pontos bons e ruins do semestre para fazer a v2 do curso.

Seria legal, inclusive, aproveitar mais o [8]. Além disso, umas aulas ficariam melhores dadas em laboratório.

Também preciso arrumar o maldito Live para rodar as coisas do pen drive sem ter de rodar um script de configuração. Isso toma um tempo chato.

Por fim e com sorte, arrumar outras referências para melhorar a qualidade do curso. Não é uma atividade que eu pretenda interromper tão cedo.

Referências Bibliográficas

[1] *Bit*, http://en.wikipedia.org/wiki/Bit, Acessado em 08/08/2013.

[2] *Caesar cipher*, http://en.wikipedia.org/wiki/Caesar_cipher, Acessado em 07/09/2013.

[3] *Crea+*, http://creamas.com.br/, Acessado em 27/11/2013.

[4] *Eniac*, http://en.wikipedia.org/wiki/ENIAC, Acessado em 01/08/2013.

[5] *Julius caesar*, http://en.wikipedia.org/wiki/Julius_Caesar, Acessado em 07/09/2013.

[6] *Pascaline*, http://en.wikipedia.org/wiki/Pascaline, Acessado em 01/08/2013.

[7] *Python (programming language)*, http://en.wikipedia.org/wiki/Python_(programming_language), Acessado em 13/09/2013.

[8] Tim Bell, Ian H. Witten e Mike Fellows, *Computer science unplugged*, csunplugged.org, 2010.

[9] Jason R. Briggs, *Python for kids: A playful introduction to programming*, no starch press, 2010.

[10] Richard Feynman, *Feynman lectures on computation*, Addison-Wesley, 1996.

[11] Python Software Foundation, *General python faq*, http://docs.python.org/2/faq/general.html, 2013, Acessado em 13/09/2013.

[12] _____, *Idle*, http://docs.python.org/2/library/idle.html, 2013, Acessado em 29/11/2013.

[13] _____, *Input and output*, http://docs.python.org/2/tutorial/inputoutput.html, 2013, Acessado em 26/09/2013.

[14] _____, *turtle — turtle graphics for tk*, http://docs.python.org/2/library/turtle.html, 2013, Acessado em 08/10/2013.

[15] Michelle A. Hoyle, *Pascaline: The first mechanical calculator*, http://www.eingang.org/Lecture/pascaline.html, 2006, Acessado em 01/08/2013.

[16] Lauri Karttunen, *Einstein's puzzle*, http://www.stanford.edu/~laurik/fsmbook/examples/Einstein%27sPuzzle.html, 2004, Acessado em 23/08/2013.

[17] Fabio Kon, Alfredo Goldman e P. J. S. Silva, *Introdução à ciência da computação com java e orientação a objetos*, IME-USP, São Paulo, 2006.

[18] Jeffrey Shallit, *A very brief history of computer science*, `https://cs.uwaterloo.ca/~shallit/Courses/134/history.html`, 1995, Acessado em 01/08/2013.

[19] Albert Sweigart, *Making games with python and pygame*, http://inventwithpython.com/pygame, 2012.

[20] www.asciitable.com, *Ascii table*, `http://www.asciitable.com/`, 2010, Acessado em 07/09/2013.