Algoritmos e Programação I: Aula 06*

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul
79070-900 Campo Grande, MS
`https://ava.ufms.br`

Sumário

1 Solucionando Problemas com Estrutura Sequencial

Vimos nas aulas anteriores que a solução de problemas usando um computador podem envolver operações matemáticas, manipulação de textos, gráficos e sons. Além disso, na soluções desses problemas podemos usar três tipos de estruturas básicas na descrição dos algoritmos/programas: sequencial, seleção e repetição. Inicialmente estamos tratando com problemas que utilizam a estrutura sequencial.

Nas aulas passadas vimos vários exemplos de problemas cujos algoritmos/programas usam a estrutura sequencial. Neste aula, vamos analisar a solução de problemas que envolvem operações matemáticas, manipulação de strings e gráficos que são um pouco mais complexas que as soluções discutidas até aqui usando a estrutura sequencial.

Como nas aulas anteriores, para projetar as soluções utilizaremos a metodologia descrita nas Aulas 03 e 04. Na descrição dos passos do algoritmo usaremos o português. Na fase da codificação, o programa correspondente ao algoritmos deve ser escrito observando a sintaxe e regras da linguagem de programação utilizada na disciplina, no nosso caso, o `Python`.

*Este material é para o uso exclusivo da disciplina de Algoritmos e Programação I da FACOM/UFMS e utiliza as referências bibliográficas da disciplina.

Como vimos anteriormente, um algoritmo para a solução de um problema, tem como entrada um conjunto de dados, executa uma sequência de passos com os dados de entrada em ordem para obter uma saída representando a solução do problema. Desta forma, um programa de computador que implementa um algoritmo deve receber e armazenar a informação dada, executar a sequência de passos necessários para solucionar o problema (os quais também podem exigir o cálculo e armazenamento de alguns passos intermediários), e finalmente imprimir os resultados. Mais especificamente, o programa descreve na linguagem de programação `Python`, como cada um dos passos do algoritmo podem ser traduzidos em instruções que possam ser executados em um computador.

Um ponto importante da metodologia utilizada na solução de problemas é a descrição do problema, durante o qual fazemos perguntas a fim de obter um entendimento preciso e completo a respeito do problema. Esse passo muitas vezes é negligenciado e acarreta soluções incorretas para o problema que está sendo solucionado. A figura abaixo ilustra a falta de um entendimento completo do problema e a pressa da implementação:

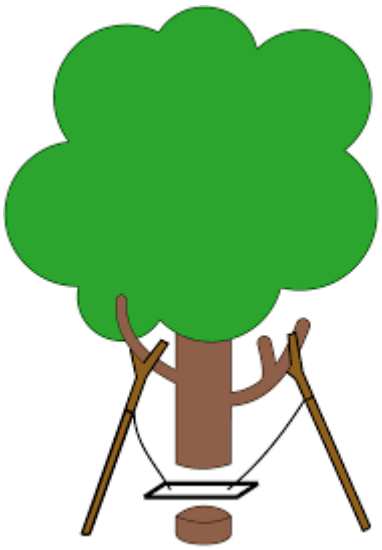

Figura 1: Importância da Descrição do Problema.

Pode parecer óbvio que não possamos resolver um problema até que não tenhamos entendido todos os detalhes, mas na maioria das vezes nem sempre é fácil clarificar esses detalhes como parece.

Nessa aula, usando a metodologia proposta, solucionaremos novos problemas algorítmicos que envolvem estrutura sequencial.

2 Solucionado Problemas Computacionais Numéricos

2.1 Horas, Minutos e Segundos

Dado uma medida de tempo em segundos, calcular e imprimir o tempo equivalente em horas, minutos e segundos.

Dado uma medida de tempo em segundos, computar e imprimir o equivalente em horas, minutos e segundos, onde minutos < 60 e segundos < 60. Nesse problema usamos uma abordagem que é utilizada em vários outros problemas. Os resultados são obtidos pela computação repetida de resto (%) e divisão (`//`). Se dividirmos o total de segundos por 60, o quociente dá o total de minutos e o resto o total de segundos que sobra. Analogamente, dividimos o total de minutos por 60. O resto é a quantidade de minutos e o quociente é o

número de horas. rePara um valor de 90 segundos, a resultado esperado é de 1 minuto (`90 // 60`) e 30 (`90 % 60`) segundos

Mesmo para um problema simples como esse, antes de continuar, mais informações a respeito do problema são necessárias para a construção do algoritmo que resolva esse problema. Necessitamos entender como o algoritmo irá ler os dados. Além disso, temos que especificar que a saída dos valores dos minutos e segundos será menor que 60 e que não haverá restrição com relação ao número de horas. Temos também que definir como será o formato da impressão dos resultados.

Em função dessas observações, o problema deve ser reescrito como:
Dado uma medida de tempo em segundos, calcular o imprimir a medida no formato horas, minutos e segundos equivalentes ao tempo em segundos, onde os valores de segundos e minutos estão entre 0 e 59, e a impressão deve atender um dado formato.

Projetaremos um algoritmo `TempoDecorrido` que computará o equivalente `horas`, `minutos` e `segundos` de um valor dado em segundos, onde os valores de `minutos` e `segundos` estarão no intervalo $[0, 59]$. Além disso, a saída da solução do problema deve ter um formato específico.
sendo que no nosso problema queremos que a saída seja no seguinte formato:
Exemplo:

```
# formato da entrada
9010 # entrada em segundos

# formato da saída
9010 Segundos = 2 Horas, 30 Minutos e 10 Segundos
```

No descrição, detalhamos os limites computacionais e matemáticos para que este algoritmo funcione corretamente. Por exemplo, precisamos nos preocupar a respeito dos casos não usuais, tais como os valores máximos e mínimos para o valor do total de segundos da entrada e os valores limites para horas, minutos e segundos da saída. Considerando que a nossa implementação será em Python, não temos a necessidade de tratar o limite do valor de entrada. Com isso, o descrição para esse problema fica:

```
# Este algoritmo lê uma medida de tempo em segundos, calcula
# o valor equivalente em horas, minutos e segundos, onde minutos e
# segundos estão no intervalo [0, 59] e imprime o resultado.
```

Após a elaboração da descrição, a próxima fase é especificar a entrada e a saída para o problema. Nessa fase, a partir de um melhor entendimento do problema e da definição mais clara do problema, desenvolvemos especificações precisas da entrada e da saída para o problema. Estas especificações definem o estado do início de qualquer execução do programa (chamadas especificações de entrada) e qual estado do programa no final da execução (chamada especificações de saída). As especificações de entrada e saída (pré e pós-condições) para o nosso problema podem ser descritas da seguinte forma:

```
# pré: (para a entrada - input)
```

e

```
pós: (para a saída - output)
```

Agora vamos identificar as `pré` (entrada) e `pós` (saída) condições para o problema de computar o equivalente em horas, minutos e segundos de uma dada medida de tempo dada em segundos.

Entrada	**Saída**
Total de Segundos `totalsegundos`	Horas, Minutos, Segundos `segundos < 60`, `minutos < 60`, `horas`

As informações de entrada e saída devem ser armazenadas na memória do computador, pois faremos operações com os valores de entrada para transformá-la na saída desejada. Como vimos no exemplo acima, os cálculos necessários para obter os segundos, minutos e horas utilizam as operações de divisão e o resto. Em função disso, necessitamos variáveis para para armazenar as essas informações.

As variáveis deste exemplo são compostas de tipos básicos. Necessitamos de `int` para armazenar os valores de `totalSegundos`, `segundos`, `minutos` e `horas`. Os identificadores abaixo representam as variáveis do tipo inteiro para armazenar as informações da entrada e saída:

```
# descrição das variáveis utilizadas
# int totalsegundos, horas, minutos, segundos
```

A solução desse problema explora os conceitos de divisão e resto. Por exemplo, como o valor equivalente em segundos é dado por um valor menor que 60, o equivalente em segundos será obtido com o resto da divisão do tempo de entrada (segundos) por 60. Se considerarmos a entrada como sendo 3661 segundos, as especificações como instruções `Python` do valor do resto da divisão, `3661 % 60 == 1`, representa o número de segundos e o valor do quociente da divisão, e `3661 // 60 == 61` representa o número de minutos da medida de tempo. Como o valor dos minutos equivalentes também deve ser dado por um valor menor que 60, repetimos o procedimento, e o equivalente em minutos será dado por `61 % 60` e o equivalente em horas por `61 // 60`.

Exemplo:

```
# formato da entrada
9010 # entrada em segundos

# estado das variáveis após a leitura
totalsegundos == 9010

# Cálculo dos segundos no intervalo [0.59]
9010 % 60 = 10
# Cálculo dos minutos no intervalo [0,59]
9010 // 60 = 150 # minutos remanescentes
(9010 // 60) % 60 = 30
# Cálculos das horas
(9010 // 60) // 60 = 2

#estado das variáveis após as operações
segundos == 10
minutos == 30
```

```
horas == 2
```

As pré e pós-condições ficam:

```
# pré: totalsegundos

# pós: horas == (totalsegundos//60)//60 and minutos ==
#      (totalsegundos//60)%60 and segundos == totalsegundos%60
```

sendo que na implementação do programa pode surgir a necessidade de utilizarmos variáveis auxiliares para armazenar valores intermediários.

Uma vez definida a entrada (`pré:`) e a saída (`pós:`), a próxima fase é o algoritmo (**subdivisão**). Nessa fase devemos descrever em passos elementares (de acordo com as funcionalidades do computador) como o problema deve ser resolvido. Lembramos que um computador pode ler e imprimir dados, realizar operações aritméticas e lógicas e controlar o fluxo dos passos (instruções).

Para o nosso exemplo temos que ler os dados, calcular a horas, minutos e segundos equivalentes a um total de segundo e imprimir os resultados. Isso pode ser descrito da seguinte forma:

```
# Algoritmo: TempoDecorrido
# Passo 1. Leia o total de segundos
# Passo 2. Calcule as horas, minutos, segundos equivalentes
# Passo 3. Imprima as horas, minutos e segundos
# fim Algoritmo
```

O algoritmo para o nosso problema deve ser expresso de acordo com as funcionalidades de um computador. No caso da leitura de um número inteiro, o Passo 1, o computador (com auxílio da linguagem de programação) tem funções específicas para a leitura de um número inteiro e armazená-lo na memória do computador. No caso da impressão, o Passo 3, os computadores (com o auxílio da linguagem de programação) também dispõem de funções para a impressão dos resultados.

O Passo 2 não parece ainda adequado para ser diretamente executado por um computador. Necessitamos fazer um refinamento (subdivisão) em sub-passos mais elementares.

```
# Passo 2. Calcule as horas, minutos, segundos
# Passo 2.1. Calcule os segundos equivalentes
# Passo 2.2. Calcule os minutos equivalentes
# Passo 2.3. Calcule as horas
```

Após a elaboração dos passos do algoritmo, passamos para a próxima fase, a finalização da codificação e documentação usando a linguagem Python. Você pode editar o programa usando o `IDLE` ou um outro ambiente de programação Python.

No Python podemos usar as funções `input()` e `print()` para entrada e saída sem necessitarmos incluir (`import`) nenhum módulo extra. Para o Passo 1, podemos ler a entrada e armazenar na variável `totalsegundos` da seguinte forma:

```
# Passo 1. Leia o total de segundos
totalsegundos = int(input())
```

e o Passo 3, a impressão deve usar uma formato específico:

```
# formato da saída
9010 Segundos = 2 Horas, 30 Minutos e 10 Segundos
```

e o Passo 3 pode ser codificado da seguinte maneira:

```
# Passo 3. Imprima as horas, minutos e segundos
print('{0:d} Segundos = {1:d} Horas, {2:d} Minutos, {3:d} Segundos.'.  \
                          format(totalsegundos, horas, minutos, segundos))
```

onde os valores da variáveis inteiras `totalsegundos`, `horas`, `minutos` e `segundos` serão colocados nas posições {0:d}, {1:d}, {2:d} e {3:d}, respectivamente.

Após o Passo 1, a informação do tempo em segundos `temposegundos` está armazenada na memória. Como o valor dos segundos pode (num problema mais complexo) ser utilizado posteriormente, é mais *elegante* computarmos segundos, minutos e horas e armazenar na memória os resultados em variáveis específicas `segundos`, `minutos` e `horas`, respectivamente. Isso pode ser feito da seguinte forma:

```
segundos =  totalsegundos % 60
minutos = (totalsegundos // 60) % 60
horas = (totalsegundos // 60) // 60
```

Utilizando o conceito de calcular e armazenar os resultados de segundos, minutos e horas em variáveis específicas, os Passos 2.1, 2.2 e 2.3 podem ser expressos da seguinte forma:

```
# Passo 2.1. Calcule os segundos equivalentes
segundos =  totalsegundos % 60
# Passo 2.2. Calcule os minutos equivalentes
totminutos = totalsegundos // 60
minutos =  totminutos % 60
# Passo 2.3. Calcule as horas
horas =  totminutos // 60
```

Para simplificar os cálculos, usamos uma variável auxiliar `totminutos`. Abaixo a codificação completa do programa em Python:

```
1  # -*- coding: utf-8 -*-
2  # Programa tempodecorrido.py
3  # Programador:
4  # Data: 22/04/2016
5  # Este algoritmo lê uma medida de tempo em segundos, calcula
6  # o valor equivalente em horas, minutos e segundos, onde minutos e
7  # segundos estão no intervalo [0, 59] e imprime o resultado.
8  # descrição das variáveis utilizadas
9  # início do módulo principal
10 # int totalsegundos, totminutos, horas, minutos, segundos
```

```
11
12  # pré: totalsegundos
13
14  # Passo 1. Leia o total de segundos
15  print('Entre com o total de segundos: ')
16  totalsegundos = int(input())
17  # Passo 2. Calcule o total de segundos, minutos e horas
18  # Passo 2.1. Calcule os segundos equivalentes
19  segundos =  totalsegundos % 60
20  # Passo 2.2. Calcule os minutos equivalentes
21  totminutos = totalsegundos // 60
22  minutos =  totminutos % 60
23  # Passo 2.3. Calcule as horas
24  horas =  totminutos // 60
25  # Passo 3. Imprima as horas, minutos e segundos
26  print('{0:d} Segundos = {1:d} Horas, {2:d} Minutos, {3:d} Segundos.'.  \
27                              format(totalsegundos, horas, minutos, segundos))
28
29  # pós: horas == totalsegundos//60*60 and minutos == (totalsegundos//60)%60
30  #      and segundos == totalSegundos%60
31  # fim do módulo principal
```

Após a edição, a compilação/interpretação é feita com o seguinte comando:

```
$ python tempodecorrido.py
```

Após a compilação (interpretação) sem erros do programa, a próximo fase é o Teste e Verificação do programa. Você pode executá-lo (de acordo com o seu sistema operacional) e testar o seu funcionamento. Nesse nosso exemplo, como só vimos a estrutura de controle sequencial, assumimos que o usuário só fornecerá um tempo total em segundos que seja válido. De qualquer forma, teste o seu programa para valores bem grandes de segundos, valores de segundos que sejam múltiplos de horas e segundos igual a 0 e verifique o que ocorre.

Abaixo, um exemplo de uma saída da execução do programa. No Python, a execução inicia juntamente com a interpretação do programa.

Exemplo:

```
# formato da entrada
Entre com o total de segundos: 9010

# formato da saída
9010 = 2 Horas, 30 Minutos, 10 Segundos.
```

No nosso exemplo do programa `tempodecorrido.py` utilizamos uma **instrução de atribuição**. Uma instrução de atribuição é utilizada para atribuir valores a variáveis e tem a seguinte forma:

```
variável = expressão
```

onde `expressão` pode ser uma constante, outra variável na qual um valor foi atribuído anteriormente, uma fórmula a ser avaliada ou a chamada para uma função. É importante lembrar que uma **instrução de atribuição** é uma **instrução de substituição**. Por exemplo, as duas instruções

```
# instrução de atribuição
a = b
b = a
```

fazem com que as variáveis `a` e `b` armazenem o mesmo valor, no caso o valor da variável `b`.

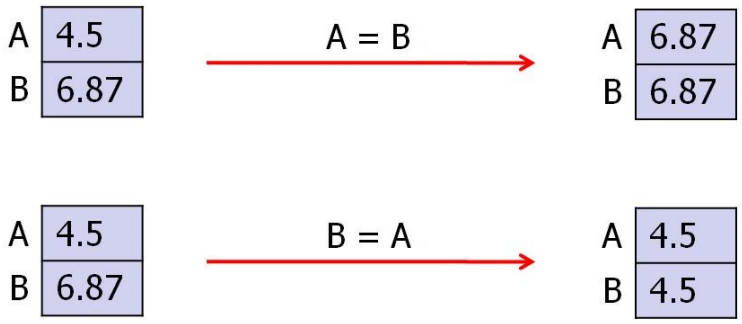

Figura 2: Exemplo de Atribuição em Python.

Como a instrução de atribuição é uma instrução de substituição, para trocarmos os valores das variáveis `a` e `b`, necessitamos de uma variável auxiliar `temp` para armazenar temporariamente o valor da variável `a`. Essa troca na linguagem `Python` também pode ser feito de uma forma mais rápida.

Imagine que você está segurando uma bola em cada uma das suas mãos e quer que a bola da mão esquerda passe para a mão direita e vice-versa. As bolas são grandes, o que impede de você conseguir segurar as duas bolas com uma única mão. As bolas também são frágeis, o que impede que elas sejam lançadas para o ar. Uma forma segura de trocar as bolas de uma mão para outra é usar um local para depositar uma das bolas, trocar a bola de mão, e com a mão que ficou vazia pegar a bola outra bola.

```
# Linguagem Python (1)
temp = a
a = b
b = temp
```

```
# Linguagem Python (2)
a, b = b, a
```

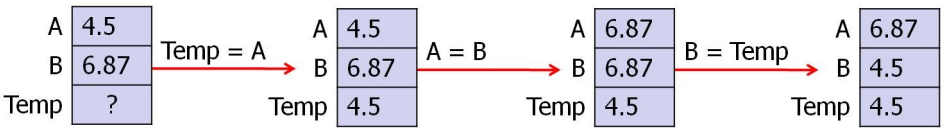

Figura 3: Exemplo de troca de valores no Python.

2.2 Media de Notas I

A avaliação da disciplina de Algoritmos e Programação é feita por meio de 3 provas e 2 trabalhos. As notas das provas e trabalhos são dadas por números de ponto flutuante com

uma casa decimal variando de 0 e 10.0. A média final é calculada somando 75% da média das provas com 25% da média dos trabalhos. A média das provas é computada pela média aritmética das notas das três provas, e a média dos trabalhos é computada pela média aritmética das notas dos dois trabalhos. As médias também são calculadas com uma casa decimal e a aproximação é feita da direita para a esquerda onde dígitos maiores ou iguais a cinco, aumentam em uma unidade o dígito da esquerda (8.445 == 8.5). Projete um algoritmo para ler as notas das três provas e dos dois trabalhos, calcular a média final de um aluno dessa disciplina e imprimir o resultado.
Exemplo:

```
# formato da entrada
7.5 # entrada da nota1
8.2 # entrada da nota2
6.4 # entrada da nota3
9.0 # entrada do trabalho1
8.0 # entrada do trabalho2

# formato da entrada
Primeira Prova    7.5
Segunda Prova     8.2
Terceira Prova    6.5
Média Provas      7.4
Primeiro Trabalho 9.0
Segundo Trabalho  8.0
Média Trabalhos   8.5
----------------------
Média Final       7.7
```

onde o cálculo das médias pode ser feito da seguinte forma:

```
Média das provas: (7.5 + 8.2 + 6.4)/3.0 = 7.4
Média dos trabalhos: (9.0 + 8.0)/2.0 = 8.5
Média Final: (0.75*7.4 + 0.25*8.5)=7.675 (aprox. 7.7)
```

Na descrição para esse problema temos que explicitar alguns detalhes do formato das notas e das aproximações que serão utilizadas nas médias.

```
# A média final de um aluno para um dado curso é computada pela soma de
# 75% da média das provas com 25% da média dos trabalhos. A média das
# provas é computada pela média aritmética das notas das três provas, e a
# média dos trabalhos é computada pela média aritmética das notas dos dois
# trabalhos. O algoritmo lê as notas das três provas e as notas dos dois
# trabalhos, computa as médias das provas, dos trabalhos e final e imprime
# o resultado. As notas são dadas por números de ponto flutuante com uma
# casa decimal variando de 0 e 10.0. As médias são calculadas também com
# uma casa decimal e a aproximação é feita da direita para a esquerda e
# dígitos maiores ou iguais a cinco, aumenta em uma unidade o dígito da
# esquerda (7.675 == 7.7).
```

As especificações da entrada e saída (domínio e imagem) ficam:

Entrada	**Saída**

5 valores de avaliação
Prova1 , Prova2, Prova3
Trabalho1, Trabalho2
`prova1`, `prova2`, `prova3`,
`trabalho1`, `trabalho2`
	Média final do aluno
	Média Provas, Média Trabalhos
	e Media Final
	`mediaprovas`, `mediatrabalhos`
	`mediafinal`

Nesse exemplo as variáveis são compostas de tipos básicos. Como as notas e as médias podem ter valores decimais, necessitamos de variáveis do tipo `float` para armazenar os valores de `prova1`, `prova2`, `prova3`, `trabalho1`, `trabalho2`, `mediaprovas`, `mediatrabalhos` e `mediafinal`. Os identificadores abaixo representam as variáveis do tipo `float` para armazenar as informações da entrada e saída:

```
# descrição das variáveis utilizadas
# float prova1, prova2, prova3, trabalho1, trabalho2
# float mediaprovas, mediatrabalhos, mediafinal
```

Quando da implementação do algoritmo pode surgir a necessidade de utilizarmos variáveis ou constantes auxiliares para armazenar valores intermediários. As pré e pós-condições ficam:

```
# pré: prova1, prova2, prova3, trabalho1, trabalhos2

# pós: mediafinal = 0.75*mediaprovas + 0.25*mediatrabalhos and
#      mediaprovas == (prova1 + prova2 + prova3) / 3.0 and
#      mediatrabalhos == (trabalho1 + trabalho2) / 2.0
```

Exemplo:

```
# formato da entrada
7.5 # entrada da nota1
8.2 # entrada da nota2
6.5 # entrada da nota3
9.0 # entrada do trabalho1
8.0 # entrada do trabalho2

# estado das variáveis após a leitura
nota1 == 7.5
nota2 == 8.2
nota3 == 6.5
trabalho1 == 9.0
trabalho2 == 8.0

# cálculo da média das provas
mediaprovas = (nota1 + nota2 + nota3)/3.0
mediaprovas = (7.5 + 8.2 + 6.5)/3.0
# cálculo da média dos trabalhos
mediatrabalhos = (trabalho1 + trabalho2)/2.0
```

```
mediatrabalhos = (9.0 + 8.0)/2.0
# cálculo da média final
mediafinal = .75*mediaprovas + .25*mediatrabalhos
mediafinal = 0.75*7.4 + 0.25*8.5

# estado das variáveis após o cálculo das médias
mediaprovas == 7.4
mediatrabalhos == 8.5
mediafinal == 7.675
```

Usando refinamentos sucessivos, os passos do algoritmo podem ser definidos como:

```
Algoritmo: Média Notas
# Passo 1. Leia as notas das provas e trabalhos
# Passo 1.1. Leia as três notas das provas
# Passo 1.2. Leia as duas notas dos trabalhos
# Passo 2. Calcule a média final da disciplina
# Passo 2.1. Calcule a média das provas
# Passo 2.2. Calcule a media dos trabalhos
# Passo 2.3. Calcule a média final
# Passo 3. Imprima o resultado
# fim do Algoritmo
```

Usando a descrição do problema, as especificações de entrada e saída e os passos do algoritmo, apresentamos abaixo uma possível codificação na linguagem Python para solucionar o problema:

```
# -*- coding: utf-8 -*-
# Programa: medianotas01.py
# Programador:
# Data: 29/03/2011
# A média final de um aluno para um dado curso é computada pela soma de
# 75% da média das provas com 25% da média dos trabalhos. A média das
# provas é computada pela média aritmética das notas das três provas, e a
# média dos trabalhos é computada pela média aritmética das notas dos dois
# trabalhos. O algoritmo lê as notas das três provas e as notas dos dois
# trabalhos, computa as médias das provas, dos trabalhos e final e imprime
# o resultado. As notas são dadas por números de ponto flutuante com uma
# casa decimal variando de 0 e 10.0. As médias são calculadas também com
# uma casa decimal e a aproximação é feita da direita para a esquerda e
# dígitos maiores ou iguais a cinco, aumenta em uma unidade o dígito da
# esquerda (7.675 == 7.7).
# início do módulo principal
# descrição das variáveis utilizadas
# float prova1, prova2, prova3, trabalho1, trabalho2,
# float mediaprovas, mediatrabalhos, mediafinal

# pré: prova1, prova2, prova3, trabalho1, trabalho2

```

```
# Passo 1. Leia as notas das provas e trabalhos
# Passo 1.1. Leia as três notas das provas
prova1 = float(input('Entre com a nota da primeira prova   : '))
prova2 = float(input('Entre com a nota da segunda prova    : '))
prova3 = float(input('Entre com a nota da terceira prova   : '))
# Passo 1.2. Leia as duas notas dos trabalhos
trabalho1 = float(input('Entre com a nota do primeiro trabalho: '))
trabalho2 = float(input('Entre com a nota do segundo trabalho : '))
# Passo 2. Calcule a média final da disciplina
# Passo 2.1. Calcule a média das provas
mediaprovas = (prova1 + prova2 + prova3) / 3.0
# Passo 2.2. Calcule a media dos trabalhos
mediatrabalhos = (trabalho1 + trabalho2) / 2.0
# Passo 2.3. Calcule a média final
mediaFinal = 0.75 * mediaprovas + 0.25 * mediatrabalhos
# Passo 3. Imprima os resultados
print('Primeira Prova    {0:.1f}'.format(prova1))
print('Segunda Prova     {0:.1f}'.format(prova2))
print('Terceira Prova    {0:.1f}'.format(prova3))
print('Média Provas      {0:.1f}'.format(mediaprovas))
print('Primeiro Trabalho {0:.1f}'.format(trabalho1))
print('Segundo Trabalho  {0:.1f}'.format(trabalho2))
print('Média Trabalhos   {0:.1f}'.format(mediatrabalhos))
print('----------------------')
print('Média Final       {0:2.1f}'.format(mediaFinal))

# pós: mediafinal = 0.75*mediaprovas + 0.25*mediatrabalhos and
#      mediaprova == (prova1 + prova2 + prova3) / 3.0 and
#      mediatrabalhos == (trabalho1 + trabalho2) / 2.0
# fim do módulo principal
```

Exemplo:

```
# formato da entrada
Entre com a nota da primeira prova   : 7.5
Entre com a nota da segunda prova    : 8.2
Entre com a nota da terceira prova   : 6.5
Entre com a nota do primeiro trabalho: 9.0
Entre com a nota do segundo trabalho : 8.0

# formato da saída
Primeira Prova    7.5
Segunda Prova     8.2
Terceira Prova    6.5
Média Provas      7.4
Primeiro Trabalho 9.0
Segundo Trabalho  8.0
Média Trabalhos   8.5
----------------------
Média Final       7.7
```

Apesar de termos especificado que as notas deveriam ser impressas com apenas uma casa decimal e que para isso deveríamos fazer uma aproximação, a nossa solução não tratou esse aspecto. As aproximações foram feitas pela função `print()`. Nesse caso, apesar de a impressão mostrar a média final como `7.7`, o valor armazenado na memória do computador é `7.675`. A aproximação só ocorreu na impressão, em função do "formatador" `{0.1f}`. No próximo exercício, faremos uma aproximação "explícita", pois se não analisarmos cuidadosamente, as operações com ponto flutuante podem acarretar erros. Numa soma de `7.35 + 7.37`, se cada um dos números for aproximado com uma casa decimal retemos `7.4 + 7.4 == 14.8`, por outro lado, se a aproximação for feita após a soma teremos `14.72 == 14.7`.

2.3 Telefone Celular Pré-Pago

Uma empresa de telefonia celular pré-pago cobra de seus clientes R$ 1.55 por cada minuto que eles utilizam no telefone para ligar para outras operadoras. Valores decimais de minutos são arredondados para o próximo número inteiro (ex. 2.45 é aproximado para 3). Ao valor total é acrescentado 28.5% de impostos. Ligações para a mesma operadora não são tarifadas. Projete e implemente um programa para ler o valor do crédito pré-existente na linha telefônica e a duração de uma chamada efetuada para uma outra operadora, o programa deve calcular o valor da chamada, o valor dos impostos, o valor total da chamada e o saldo dos créditos da linha após a ligação.

A entrada é dada por dois números reais, o primeiro indicando o crédito existente e o segundo a duração da chamada a ser contabilizada. A saída consiste em imprimir o valor do crédito existente, o valor da chamada, o valor dos impostos, o valor total da chamada (valor da chamada + impostos) e o valor do saldo remanescente após a ligação, com mensagens apropriadas, seguido de uma nova linha. Os valores devem ser impressos com um espaço após o símbolo do valor em reais (`R$`) e com três casas para inteiros e duas casas decimais. O programa deve cuidar do arredondamento do saldo, ou seja R$ 7.5650 deve ser aproximado para R$ 7.57.

Exemplo:

```
# formato da entrada
22.78 # entrada do valor do saldo
3.45 # entrada do tempo da chamada

# formato da saída
Valor dos Créditos     :R$  22.78
Valor da Chamada       :R$   6.20
Valor dos Impostos     :R$   1.15
Valor Total da Chamada :R$   7.35
Valor do Saldo Atual   :R$  15.43
```

A descrição abaixo apresenta mais alguns detalhes da definição do problema:

```
# Este programa calcula a saldo de um telefone pré-pago. O programa lê o
# crédito pré-existente e a duração da chamada, calcula o custo da
# chamada, o valor dos impostos, o valor total da chamada e o saldo
# atual. O programa imprime os resultados do custo da chamada, do custo
```

```
# dos impostos, do custo do total da chamada e do saldo atual. O valor do
# saldo atual deve ser arrendondado com duas casas decimais. Os valores
# do custo por minuto e da taxa dos impostos são dados por constantes
```

A seguir apresentamos as especificações de entrada e saída para o problema (domínio e imagem):

Entrada	**Saída**
Alíquota do imposto, custo do minuto, Saldo Existente e Duração da Chamada `MINUTO`, `IMPOSTO`, `saldo_anterior` e `duracao`	Custo da Ligação, valor impostos, custo total da ligação e Saldo Atual `valor_chamada`, `valor_impostos`, `valor_totalchamada` e `saldo_atual`

Nesse exemplo as variáveis são compostas de tipos básicos para armazenar as variáveis. Os identificadores abaixo representam as variáveis e constantes do tipo `float` para armazenar as informações da entrada e saída:

```
# descrição das constantes e variáveis utilizadas
MINUTO = 1.55 # valor do minuto
IMPOSTO = 28.5 # alíquota de imposto
# float saldo_anterior, duracao, valor_chamada
# float valor_impostos, valor_totalchamada, saldo_atual
```

Quando da implementação do algoritmo pode surgir a necessidade de utilizarmos variáveis ou constantes auxiliares para armazenar valores intermediários. Para obter o valor do minuto cheio, você pode utilizar a função/método teto `ceil(x)` da biblioteca `math` na linguagem *Python*. Uma das formas de usar a função `ceil(x)` é `math.ceil(x)`. Na implementação detalharemos como incluir a biblioteca `math`.

Exemplo:

```
# formato da entrada
22.78 # entrada do valor do saldo
3.45 # entrada do tempo da chamada

# estado das variáveis após a leitura
saldo_anterior == 22.78
duracao == 3.45

# cálculo do minuto cheio
duracao = math.ceil(duracao)
duracao = math.ceil(3.45)
# cálculo do valor da chamada sem impostos
valor_chamada = duracao*MINUTO
valor_chamada = 4*1.55
# cálculo dos impostos
valor_impostos = valor_chamada * IMPOSTO
valor_impostos = 6.2*0.285
# cálculo do valor total da chamada
valor_totachamada = valor_chamada + valor_impostos
valor_totalchamada = 6.2 + 1.767
```

```
# cálculo do saldo atual
saldo_atual = saldo_anterior - valor_totalchamada
saldo_atual = 22.78 - 7.967

# estado das variáveis após o cálculo das médias
duracao == 4
valor_chamada == 6.2
valor_impostos == 1.767
valor_totalchamada == 7.967
saldo_atual == 14.813
```

Abaixo um exemplo de uma estratégia de como um arredondamento explicito com duas casas decimais pode ser feito:

```
# Arredondamento com duas casas decimais em Python
saldo_atual = saldo_anterior - valor_totalchamada
temp = int(100.0 * saldo_atual + 0.5)
saldo_atual = temp/100.00
```

As variáveis `saldo_atual`, `saldo_anterior`, `valor_totalchamada` e `saldo_atual` armazenam números com ponto flutuante, e `temp` é uma variável que armazena um número inteiro. A atribuição

```
temp = int(100.0 * saldo_atual + 0.5)
```

multiplica o valor de `saldo_atual` por `100.0` e move o ponto flutuante para depois da segunda casa decimal:

```
2.0555*100.0 == 205.55
```

ao somarmos `0.5` garantimos que os valores da primeira casa decimal (terceira casa decimal no valor original de `saldo_atual`) `>= 5` aumentarão em um (1) o valor da unidade, e para valores da primeira casa `< 5` não alterarão o valor da unidade.

```
205.55 + 0.5 == 206.05
```

Usando o *cast* (`int`), garantimos que a parte do ponto flutuante será desconsiderada:

```
int(206.05) == 206
```

Ao dividirmos por `100.00`, retomamos o valor inicial de `saldo_atual`, com o arredondamento desejado, e com a garantia de que apenas duas casas decimais estarão sendo consideradas (as demais casas decimais agora são iguais a zero):

```
206/100.0 == 2.06
```

Após a descrição e as especificações de entrada e saída, as pré e pós-condições ficam:

```
# pré: MINUTO, IMPOSTO, saldo_anterior, duracao

# pós: saldo_atual = saldo_anterior - valor_totalchamada and
#      valor_totalchamada == valor_chamada + valor_impostos and
#      valor_chamada == math.ceil(duracao)*MINUTO and
#      valor_impostos = valor_chamada*IMPOSTOS
```

Temos que descrever os passos do algoritmo que transformam a entrada na saída do problema. Numa primeira abordagem, os passos do algoritmo ficam:

```
# Algoritmo: Pré-Pago
# Passo 1. Leia o saldo anterior e a duração da chamada
# Passo 2. Calcule valor chamada, impostos e total
# Passo 3. Imprima os resultado
# fim do Algoritmo
```

O Passo 2 necessita de um melhor detalhamento, pois da forma como ele está descrito não pode ser implementado por uma instrução num computador. Usando refinamentos sucessivos obtemos:

```
# Passo 2. Calcule valor chamada, impostos e total
# Passo 2.1. Calcule o valor do minuto cheio
# Passo 2.2. Calcule o custo da chamada
# Passo 2.3. Calcule o valor dos impostos
# Passo 2.4. Calcule o valor total da chamada
# Passo 2.5. Calcule o arrendondamento do valor total da chamada
```

Como vimos anteriormente, o Passo 2.1. pode sr implementado com a função `ceil(x)` da biblioteca `math`. Para isso, temos que no início do programa "importar" a bilioteca `math`:

```
# declaração das bibliotecas utilizadas
import math
```

Na implementação do Passo 3, os valores devem ser impressos com 7 espaços, sendo duas casas decimais. O (.) também ocupa um espaço e para obter esse formato dê uma lida no material sobre a função `print` ({`0:7.2f`}). Os valores devem ser precedidos do símbolo `R$`.

Após a descrição, as especificações de entrada e saída e as informações acima, descrevemos uma implementação da solução do problema na linguagem Python. Você pode editar o programa usando o `IDLE` ou um ambiente de programação Python.

```
1 # -*- coding: utf-8 -*-
2 # Programa: telefone.py
3 # Programador:
4 # Data: 03/04/2011
5 # Este programa calcula a saldo de um telefone pré-pago. O programa lê o
6 # crédito pré-existente e a duração da chamada, calcula o custo da
7 # chamada, o valor dos impostos, o valor total da chamada e o saldo
8 # atual. O programa imprime os resultados do custo da chamada, do custo
```

```
# dos impostos, do custo do total da chamada e do saldo atual. O valor do
# saldo atual deve ser arrendondado com duas casas decimais. Os valores
# do custo por minuto e da taxa dos impostos são dados por constantes
# início do módulo principal
# declaração das bibliotecas
import math
# descrição das constantes e variáveis utilizadas
MINUTO = 1.55 # valor do minuto
IMPOSTO = 28.5 # alíquota de imposto
# float saldo_anterior, duracao, valor_chamada
# float valor_impostos, valor_totalchamada, saldo_atual

# pré: saldo_anterior, duracao

# Passo 1. Leia o saldo anterior e a duração da chamada
# Passo 1.1. Leia o saldo anterior
saldo_anterior = float(input('Leia o crédito pré-existente :R$ '))
# Passo 1.2. Leia a duração da chamada
duracao = float(input('Leia a duração da chamada    : '))
# Passo 2. Calcule valor chamada, impostos e total
# Passo 2.1. Calcule o valor do minuto cheio
duracao = math.ceil(duracao)
# Passo 2.2. Calcule o custo da chamada
valor_chamada = duracao*MINUTO
# Passo 2.3. Calcule o valor dos impostos
valor_impostos = (valor_chamada * IMPOSTO)/100.0
# Passo 2.4. Calcule o valor total da chamada
valor_totalchamada = valor_chamada + valor_impostos
# Passo 2.5. Calcule o arrendondamento do valor total da chamada
saldo_atual = saldo_anterior - valor_totalchamada
temp = int(100.0 * saldo_atual + 0.5)
saldo_atual = temp/100.0
# Passo 3. Imprima os resultado
print('Valor dos Créditos     :R${0:7.2f}'.format(saldo_anterior))
print('Valor da Chamada       :R${0:7.2f}'.format(valor_chamada))
print('Valor dos Impostos     :R${0:7.2f}'.format(valor_impostos))
print('Valor Total da Chamada :R${0:7.2f}'.format(valor_totalchamada))
print('Valor do Saldo Atual   :R${0:7.2f}'.format(saldo_atual))

# pós: saldo_Anterior valor_chamada valor_impostos valor_totalchamadas
#      saldo_atual
# fim do módulo principal
```

Exemplo:

```
# formato da entrada
22.78
3.45

# formato da saída
```

```
Valor dos Créditos     :R$  22.78
Valor da Chamada       :R$   6.20
Valor dos Impostos     :R$   1.15
Valor Total da Chamada :R$   7.35
Valor do Saldo Atual   :R$  15.43
```

3 Solucionando Problemas Computacionais com Texto

3.1 Contando Palavras em um Texto

Os tipos de dados numéricos `int` e `float` são usados na grande maioria dos exemplos apresentados até agora. Contudo, existe uma grande classe de problemas que não pode ser resolvida somente com esses tipos. Uma dessas classes é de problemas que necessitam a manipulação e processamento de textos e para serem resolvidos. Para isso, necessitamos de um tipo de dados que armazene conjuntos de caracteres. Geralmente, esse problemas tratam com conjuntos de caracteres ASCII denominados *strings*, e suas soluções requerem além do uso de um tipo de dados específico para armazenar strings, a utilização de um conjunto de operadores que possam manipular variáveis deste tipo. Na nossa disciplina, definimos uma **string** como:
Definição - Uma **string** é uma sequência finita de zero ou mais caracteres ASCII. Quando utilizadas em um programa Python devem estar entre dois apóstrofos (') (no Pythom também pode ser usado aspas ("), para distingui-las das instruções do programa.

Para armazenar uma **string** usando a linguagem `Python`, necessitamos de n posições na memória - para os n caracteres que a compõe. Por exemplo, entrada

```
'Entre uma lista de notas'
```

é uma string.

A manipulação de strings varia muito em função da linguagem de programação. O Python tem um conjunto poderoso de métodos e funções para manipulação de strings que descreveremos no decorrer da disciplina. Como o Python usa tipagem dinâmica, da mesma forma que com as variáveis numéricas, não necessitamos declarar variáveis do tipo string (`str`).

De posse dessas definições, voltemos ao problema de ler uma string qualquer `s` e computar e imprimir o número de palavras existente nesta string.
Exemplo:

```
# formato da entrada
palavras, palavras e nada mais # string da entrada

# formato desejado de saída
5
```

No fase da descrição das especificações para resolver este problema, temos que definir como será a entrada e o que entendemos por uma palavra. A entrada será dada por uma linha de caracteres ASCII visíveis (não pode ter caracteres de controle). Consideraremos uma palavra como uma sequência contígua qualquer de caracteres ASCII visíveis sem o

espaço (o espaço será o delimitador das palavras na linha de entrada). Por simplicidade, vamos assumir que os sinais de pontuação, tais como ponto, vírgula e assim por diante, também são considerados como parte da palavra nas quais eles aparecem. Desta forma, cada uma das seguintes três strings:

```
'palavras, palavras e nada mais'
'Aula de Algoritmos!'
'Aula.'
```

é formada pelas palavras, respectivamente:

```
'palavras,' 'palavras' 'e' 'nada' 'mais'
'Aula' 'de' 'Algoritmos!'
'Aula.'
```

A descrição especifica mais alguns detalhes da especificação do problema.

```
# Este programa lê uma linha de caracteres ASCII visíveis (sem caracteres
# de controle, computa o número de palavras na linha de entrada (conjunto
# de caracteres ASCII visíveis separados por um caractere espaço) e
# imprime o resultado. Os sinais de pontuação são considerados como parte
# das palavras.
```

Uma vez definido claramente o que será a string de entrada e o que consideraremos como palavra, as especificações de entrada e saída (domínio e imagem) para o problema ficam:

Entrada	**Saída**
Uma linha de texto ASCII sem caracteres de controle `palavras`	Número de palavras no texto de entrada `numpalavras`

Como ainda não sabemos manipular strings e só conhecemos a estrutura de controle sequencial, vamos resolver esse problema armazenado as palavras (string) do texto de entrada numa lista (`list`). Posteriormente veremos mais detalhes dessa estrutura de dados. Além disso, utilizaremos a função/método `len()` para computar o número de elementos da lista. Os identificadores abaixo representam as variáveis do tipo `list` e `int` para armazenar as informações da entrada e saída:

```
# descrição das variáveis utilizadas
# list palavras
# int numpalavras
```

Exemplo:

```
# formato da entrada
palavras, palavras e nada mais # entrada das palavras

# estado das variáveis após a leitura
palavras = ['palavras,', 'palavras', 'e', 'nada', 'mais']
```

```
# cálculo do número de palavras
numpalavras = len(palavras)

# estado das variáveis após o cálculo do número de palavras
numpalavras == 5
```

Após uma descrição mais detalhada do problema, e as especificações da entrada e saída, podemos detalhar as pré e pós-condições para o problema de contar palavras. O detalhamento abaixo parece ser apropriado:

```
# pré: uma série de caracteres c[0]c[1]...c[n] and n >= 0
#      and para todo i em {0,...,n}: c[i] em ASCII visível

# pós: numpalavras == número de palavras do texto de entrada
```

Nesse detalhamento, usamos uma mistura de lógica e Português para expressar as pré e pós-condições para esse problema. Inicialmente, as pré-condições expressam somente que a entrada é um fluxo de caracteres ASCII visíveis. Vamos agora descrever uma sequência de passos do algoritmo que soluciona o problema.

```
Algoritmo: Conta Palavras
# Passo 1. Leia uma string
# Passo 2. Calcule o número de palavras da string
# Passo 3. Imprima o número de palavras
# fim do Algoritmo
```

Após a descrição dos passos do algoritmo, passamos para a Codificação na linguagem Python. Vamos descrever como podemos implementar em Python os Passos 1 e 2. A leitura de umacstring no Python pode ser feita como:

```
# Passo 1. Leia uma string
palavras = input()
```

onde um conjunto de caracteres é atribuído à variável `palavras`. No caso da entrada `'aulas segundas, quartas e sextas'`, o estado da variável `palavras` após a leitura fica:

```
palavras == 'aulas segundas, terças e sextas'
```

Como queremos computar o número de palavras da string, e ainda não dispomos de instruções de seleção e repetição, vamos usar outros recursos da linguagem Python para solucionar esse problema. O método `string.split(separador, max)` divide uma string numa lista (`list`) de `max` substrings, usando o `separador`. Se omitirmos os parâmetros `separador` e `max`, o método divide a string de entrada em uma lista de substrings usando o caractere espaço.

```
palavras = palavras.split()

# estado da variável palavras após a instrução acima
palavras == ['aulas', 'segundas,', 'terças', 'e', 'sextas']
```

e em função disso, podemos reescrever o Passo 1 da seguinte maneira:

```
# Passo 1. Leia uma string e separe as palavras
palavras = input().split()
```

No caso do Passo 2 podemos usar a função `len()` que computa o número de elementos de um objeto. Já usamos essa função para computar o número de caracteres de uma string.

```
# Passo 2. Compute o número de palavras da lista
tam = len(palavras)
```

Vimos que em função das características do Python, fizemos alguns refinamentos nos passos do algoritmo. Após esses refinamentos, passamos para a fase da finalização da codificação. Uma possível implementação em Python da solução desse problema é dado pelo programa `palavras.py`. Você pode editar o programa usando o `IDLE` ou um outro ambiente de programação para Python.

```
1  # -*- coding: utf-8 -*-
2  # Programa: palavras.py
3  # Programador:
4  # Data: 26/04/2012
5  # Este programa lê uma linha de caracteres ASCII visíveis (sem caracteres
6  # de controle, computa o número de palavras na linha de entrada (conjunto
7  # de caracteres ASCII visíveis separados por um caractere espaço) e
8  # imprime o resultado. Os sinais de pontuação são considerados como parte
9  # das palavras.
10 # início do módulo principal
11 # descrição das variáveis utilizadas
12 # list palavras
13 # int numpalavras
14
15 # pré: uma série de caracteres c[0]c[1]...c[n] and n >= 0
16 #      and para todo i em {0,...,n}: c[i] em ASCII visível
17
18 # Passo 1. Leia uma string e separe as palavras
19 palavras = input().split()
20 # Passo 2. Compute o número de palavras da lista
21 tam = len(palavras)
22 # Passo 3. Imprima o número de palavras
23 print('{0:d}'.format(tam))
24
25 # pós: numpalavras == número de palavras do texto de entrada
26 # fim do módulo principal
```

Exemplo:

```
# formato da entrada
palavras, palavras e nada mais

# formato da saída
5
```

4 Solucionando Problemas Computacionais com Ambiente Gráfico

Embora os tipos de dados numéricos e strings atendem a uma grande variedade de problemas algorítmicos, eles não resolvem problemas que requerem a representação de informações gráficas. Tais problemas incluem a impressão e análise de raios x, imagens de tomografia computadorizada, imagens de satélites, jogos, e vários gráficos e diagramas que são usados frequentemente em negócios, computação científica, etc.

Nesta aula iniciamos a implementação de um Jogo da Velha. Na medida que as estruturas de seleção e repetição forem discutidas, ampliaremos o escopo da implementação. Inicialmente, apenas desenharemos o tabuleiro do jogo.

4.1 Jogo da Velha - I

Nas aulas anteriores, tratamos problemas de imprimir pontos, retas e figuras na tela do computador. Vamos agora ilustrar a solução algorítmica do problema do jogo da velha. Inicialmente vamos apenas imprimir um reticulado na tela com um mensagens, mas no decorrer da disciplina vamos ampliar o nosso algoritmo para monitorar os movimentos individuais de um jogo da velha, imprimindo os resultados de cada movimento ("X" ou "O") em um reticulado retangular na tela do computador.

A descrição, devemos estabelecer a natureza da entrada e saída para este problema. Nesse primeiro exemplo queremos imprimir dois "movimentos" no jogo, (X e O) que serão indicados por um par de floats que indicam a posição no reticulado onde "X" ou "O" devem ser colocados (Veja Figura 5). A saída deve ser a impressão do tabuleiro com os dois movimentos (Figura 5).

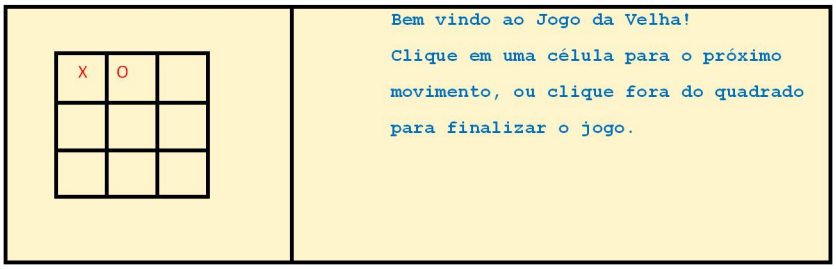

Figura 4: Monitorando o Jogo da Velha.

```
# Este programa utiliza a biblioteca PIL. Ele gera e imprime um
# reticulado 3 x 3. O programa imprime dois movimentos, um para
# X e outro para O, e imprime as informações do jogo na tela.
```

Após uma descrição mais detalhado do problema, as especificações de entrada e saída ficam:

Entrada	**Saída**
Dois pares de coordenadas um para cada movimento `x1,y1, x2, y2`	Um tabuleiro com um reticulado 3 x 3 para o jogo da velha e a impressão de dois movimentos.

Os identificadores abaixo representam as variáveis do tipo `float` para armazenar as informações da entrada e saída:

```
# descrição das variáveis e constantes utilizadas
# float x1, y1, x2, y2
```

Com a descrição mais detalhada do problema e as especificações de entrada e saída, podemos descrever as pré e pós-condições para o problema. Note que as especificações para este problema são estabelecidas mais informalmente que as dos problemas anteriores.

```
# pré: x1, y1, x2, y2, dimensões e posição do tabuleiro e
#      reticulado e conteúdo das mensagens.

# pós: Um tabuleiro com um jogo da velha e a impressão de
#      dois movimentos.
```

A subdivisão do nosso problema fica:

```
# Passo 1. Defina o ambiente gráfico
# Passo 1.1. Defina o tamanho da imagem/tela
# Passo 1.2. Defina o padrão de cores
# Passo 1.3. Crie a imagem/tela
# Passo 1.4. Atribua as coordenadas dos movimentos
# Passo 2. Desenhe um tabuleiro para o jogo da velha
# Passo 2.1. Gere um retângulo de fundo laranja
# Passo 2.2. Gere um reticulado 3 x 3
# Passo 2.2.1. Desenhe as linhas horizontais internas
# Passo 2.2.2. Desenhe as linhas verticais internas
# Passo 2.3. Gere as mensagens
# Passo 2.3.1. Desenhe uma linha vertical separando o grid
# Passo 2.3.1. Defina o tipo e tamanho de letra para as mensagens
# Passo 2.3.2. Gere as mensagens
# Passo 2.4. Atribua os movimentos
# Passo 2.4.1. Defina o tipo e tamanho de letra para os movimentos
# Passo 2.4.2. Gere os movimentos
# Passo 3. Imprima o jogo na tela
```

A subdivisão acima usa as funcionalidades dos módulos `Image`, `ImageDraw` e `ImageFont` da biblioteca `PIL`. Uma implementação em `Python` fica:

```
1  # -*- coding: utf8 -*-
2  # Programa: jogodavelha00pil.py
3  # Programador:
4  # Data: 23/08/2020
5  # Este programa utiliza a biblioteca PIL. Ele gera e imprime um
6  # reticulado 3 x 3. O programa imprime dois movimentos, um para
7  # X e outro para O, e imprime as informações do jogo na tela.
8  # declaração das bibliotecas utilizadas
9  from PIL import Image, ImageDraw, ImageFont
10 # início do módulo principal
11 # Descrição das variáveis e constantes utilizadas
```

```
12 # float x1, y1, x2, y2
13 # str padrao
14 # Image tela
15 # ImageDraw desenha
16
17 # pré: x1, y1, x2, y2, dimensões e posição do tabuleiro e
18 #      reticulado e conteúdo das mensagens.
19
20 # Passo 1. Defina o ambiente gráfico
21 # Passo 1.1. Defina o tamanho da imagem/tela
22 largura = 800
23 altura = 500
24 # Passo 1.2. Defina o padrão de cores
25 padrao = 'RGB'
26 # Passo 1.3. Crie a imagem/tela
27 tela = Image.new(padrao, (largura,altura), color=(255,150,0))
28 desenhe = ImageDraw.Draw(tela)
29 # Passo 1.4. Atribua as coordenadas dos movimentos
30 x1, y1 = 0.1, 0.75
31 x2, y2 = 0.2, 0.75
32 # Passo 2. Desenhe um tabuleiro para o jogo da velha
33 # Passo 2.1. Gere um retângulo de fundo laranja
34 laranja = (255,200,0)
35 desenhe.rectangle((100,100,400,400),outline='black',fill=laranja,width=3)
36 # Passo 2.2. Gere um reticulado 3 x 3
37 # Passo 2.2.1. Desenhe as linhas horizontais internas
38 desenhe.line((100, 200, 400, 200),fill=(0, 0, 0), width=3)
39 desenhe.line((100, 300, 400, 300),fill=(0, 0, 0), width=3)
40 # Passo 2.2.2. Desenhe as linhas verticais internas
41 desenhe.line((200, 100, 200, 400),fill=(0, 0, 0), width=3)
42 desenhe.line((300, 100, 300, 400),fill=(0, 0, 0), width=3)
43 # Passo 2.3. Gere as mensagens
44 # Passo 2.3.1. Desenhe uma linha vertical separando o grid
45 desenhe.line((450, 0, 450, 500),fill=(0, 0, 0), width=3)
46 # Passo 2.3.1. Defina o tipo e tamanho de letra para as mensagens
47 fnt = ImageFont.truetype('arial', 20)
48 # Passo 2.3.2. Gere as mensagens
49 desenhe.text((500,100),'Bem Vindo ao Jogo da Velha!',font=fnt,fill=(0,0,0))
50 desenhe.text((500,140),'Clique numa célula para o',font=fnt,fill=(0,0,0))
51 desenhe.text((500,180),'próximo movimento, ou clique',font=fnt,fill=(0,0,0))
52 desenhe.text((500,220),'fora do reticulado para',font=fnt,fill=(0,0,0))
53 desenhe.text((500,260),'finalizar o jogo.',font=fnt,fill=(0,0,0))
54 # Passo 2.4. Atribua os movimentos
55 # Passo 2.4.1. Defina o tipo e tamanho de letra para os movimentos
56 fntm = ImageFont.truetype('arial', 40)
57 # Passo 2.4.2. Gere os movimentos
58 desenhe.text((135,130),'X',font=fntm,fill=(255,0,0))
59 desenhe.text((235,130),'O',font=fntm,fill=(255,0,0))
60 # Passo 3. Imprima o jogo na tela
61 tela.show()
```

```
62
63 # pós: Um tabuleiro com um jogo da velha e a impressão de
64 #      dois movimentos.
65 # fim do módulo principal
```

Edite o programa e execute. A saída deve ser como:

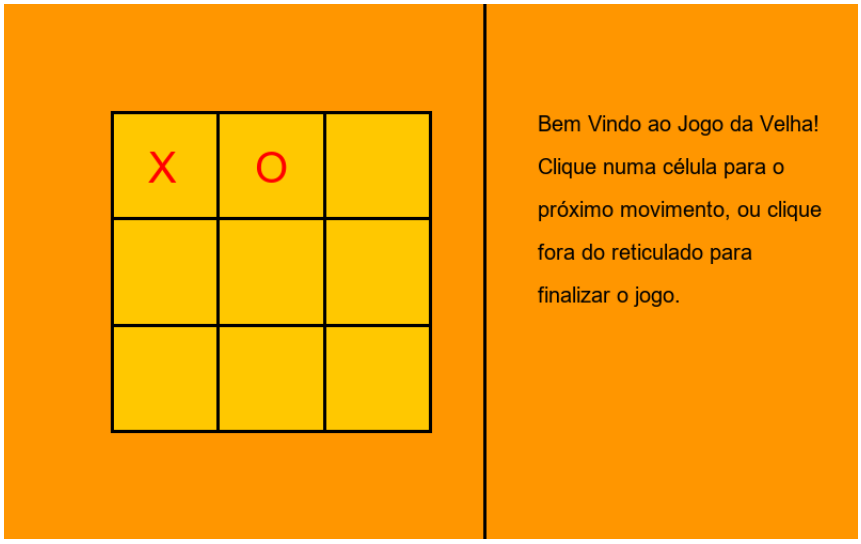

Figura 5: Monitorando o Jogo da Velha.

Posteriormente teremos que abordar algumas outras questões a respeito do jogo. Temos que definir se o jogo é para um ou dois jogadores. Se for para um jogador, o computador será dotado de alguma estratégia para jogar? Os movimentos serão inseridos por coordenadas ou por *cliques* com o mouse? O primeiro jogador será `X` ou `O`? Como será indicado que um jogador ganhou?

5 Observações

Nos problemas tratados na disciplina de Algoritmos e Programação I, sempre precisaremos conhecer todas as informações necessárias na declaração do problema antes de iniciar as Fase 2 a 6 da metodologia utilizada. Não é incomum ficarmos tentados a saltar diretamente a uma dessas fases - muitas vezes começamos a codificar antes analisar um pouco mais o problema, e logo em seguida descobrir que necessitamos mais informações antes de ir adiante, sendo muitas vezes necessário retornar a uma fase anterior. Em computação deve-se evitar esta tentação tanto quanto possível. Isto é, devemos completar cada fase da metodologia antes de mover-nos para a próxima fase, mesmo que o problema que está sendo resolvido seja bem simples.

Por outro lado, muitas vezes não possível obter todas as informações a respeito do problema e sua solução antes de passarmos para as fases posteriores da metodologia. Desta forma seria muito simplista pensar que a metodologia sempre é uma sequência linear de fases; muitas vezes um estado inicial é repetido após alguma experiência tenha sido adquirida nos estágios posteriores do desenvolvimento da solução. A metodologia que usamos ilustra bem o caso da estratégia do modelo da queda d´água.