Algoritmos e Programação I: Aula 10*

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

Sumário

1 Usando a Estrutura de Seleção com mais de uma via

Vimos na aula anterior que a estrutura de controle condiciona faz possível a seleção e alteração na sequência de execução das instruções de um programa. Os problemas que abordamos avaliavam uma expressão lógica, e se verdadeira, o programa executava uma (ou um bloco) instrução e desviava o fluxo para depois da instrução (ou bloco) do comando `else:`, ou se falsa, ignorava a (ou o bloco) instrução entre o `:` e o `else:`, executava uma (ou um bloco) de instruções após o comando `else:` e continuava com o fluxo do programa.

Essa estrutura é adequada para solucionar problemas onde as decisões são tomadas em função da avaliação `True` ou `False` da expressão lógica. Por exemplo, considere o problema de computar uma mensagem `'AP'` caso a média final do aluno seja maior ou igual a 6.0 ou `'RP'` caso contrário:

```
if media >= 6.0:
   msg = 'AP'
else:
   msg = 'RP'
```

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

Na aula de hoje vamos considerar uma estrutura de seleção mais complexa na qual existem mais de duas possibilidades para definir o fluxo de ações. Esse caso pode ser ilustrado pelo problema de computar a mensagem para um aluno, dependendo da frequência e nota do estudante. Para um aluno ser considerado aprovado numa disciplina ela precisa ter média maior ou igual a 6.0 e frequência maior ou igual a 75%. Caso o aluno não tenha frequência maior ou igual a 75%, ele recebe uma menção `'RF'`, reprovado por falta e é considerado reprovado independente da nota da média (mesmo que a média seja maior ou igual a 6.0). Caso ele tenha frequência maior ou igual a 75%, será analisado a média do aluno. Se a média for maior ou igual a 6.0, o aluno recebe a menção `'AP'`, aprovado, caso contrário recebe a menção `'RN'`, reprovado por nota. Isto pode ser ilustrado com o seguinte diagrama da Figura 1.

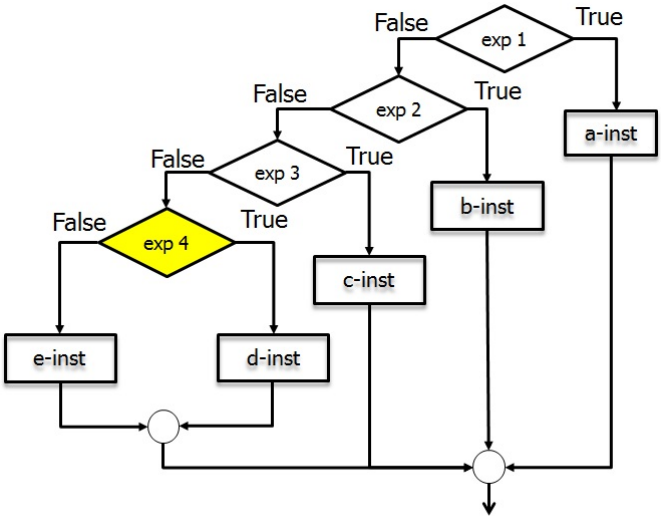

Figura 1: Estrutura de Seleção `if elif`.

Esta estrutura de seleção pode ser implementada em *Python* usando uma instrução `if`, `elif` e `else` da seguinte forma:

```
1  if expressão_booleana_1: # True vai para linha 2 - False vai para linha 6
2     a_instrução-1 # vai p/ linha 3
3     a_instrução-2 # vai p/ linha 4
4     ...
5     a_instrução-n # vai p/ instrução após a linha 15
6  elif expressão_booleana_2: # True vai p/ linha 7 - False vai p/ linha 11
7     b_instrução-1 # vai p/ linha 8
8     b_instrução-2 # vai p/ linha 9
9     ...
10    b_instrução-m # vai p/ instrução após a linha 15
11 else: # vai p/ linha 12
12    d_instrução-1 # vai p/ linha 13
13    d_instrução-2 # vai p/ linha 14
14    ...
15    d_instrução-p # vai p/ instrução após a linha 15
```

Se a expressão booleana `expressão_booleana_1` for verdadeira (`True`), as instruções do bloco `a_instrução-1 .. a_instrução-n` serão executadas e as sequências de instruções

dos blocos b_instrução-1 .. b_instrução-m e c_instrução-1 .. c_instrução-p serão ignoradas.

Caso contrário, a sequência de instruções-1 será ignorada e e será avaliada a expressão lógica b_expressão_booleana_2. Se b_expressão_booleana_2 for verdadeira (True), as instruções do bloco b_instrução-1 .. b_instrução-m serão executadas e as instruções do bloco c_instrução-1 .. c_instrução-p serão ignoradas.

No caso em que as expressão_booleana_1 e b_expressão_booleana_2 forem falsas, os blocos a_instrução-1 .. a_instrução-n e b_instrução-1 .. b_instrução-m serão ignoradas e as instruções do bloco c_instrução-1 .. c_instrução-p serão executadas.

Em qualquer caso, a execução continua com a instrução seguindo o final do bloco `if elif else` a menos que a execução seja finalizada ou transferida para uma outra parte do programa por uma das instruções na sequência de instruções selecionada. A estrutura de seleção `if elif else` permite ao programador não somente especificar a sequência de instruções selecionadas para a execução quando a expressão booleana for `True` (`verdadeira`) mas também para indicar várias blocos de instruções alternativas para a execução quando ela for `False` (`falsa`). No exemplo

```
1 if frequencia < 75: # True vai para linha 2 - False vai para linha 3
2    msg = 'RF' # vai p/ instrução após a linha 6
3 elif media >= 6.0: # True vai p/ linha 4 - False vai para linha 5
4    msg = 'AP' # vai p/ instrução após a linha 6
5 else: # vai p/ linha 6
6    msg = 'RN' # vai p/ instrução após a linha 6
```

onde a expressão booleana frequencia < 75 é avaliada, se ela for verdadeira será feita a atribuição msg = 'RF' e o programa será desviado para a primeira instrução após a linha 6. Caso ela seja falsa, o programa será desviado para a linha 3, e o comando elif que será executado. A expressão booleana media >= 6.0 será avaliada e se ela for verdadeira, a atribuição 'AP' será feita e o programa será desviado para a primeira instrução após a linha 6. Se a expressão booleana media >= 6.0 for falsa, o programa será desviado para a linha 5 e será atribuído a mensagem 'RN'. Em qualquer caso, ao final de uma dessas atribuições, a execução do programa continua com a instrução seguinte ao final da indentação do bloco `if elif else` (após a linha 6).

Essa mesma solução também pode ser obtida com a combinação de comandos `if`'s.

```
1 if expressão booleana 1: # True vai p/ linha 2 - False vai p/ linha 7
2    sequência de instruções-a # vai p/ linha 3
3    if expressão booleana 2: # True vai p/ linha 4 - False vai p/ linha 5
4       sequência de instruções-b # vai p/ instrução após a linha 8
5    else: # vai p/ linha 6
6       sequência de instruções-c # vai p/ instrução após a linha 8
7 else: # vai p/ linha 8
8    sequência de instruções-d # vai p/ instrução após a linha 8
```

```
1 if frequencia >= 75: # True vai para linha 2 - False vai para linha 6
2    if media >= 6.0: # True vai p/ linha 3 - False vai p/ linha 4
3       msg = 'AP' # vai p/ instrução após a linha 7
4    else: # vai p/ linha 5
5       msg = 'RN' # vai p/ instrução após a linha 7
6 else: # vai p/ linha 7
7    msg = 'RF' # vai p/ instrução após a linha 7
```

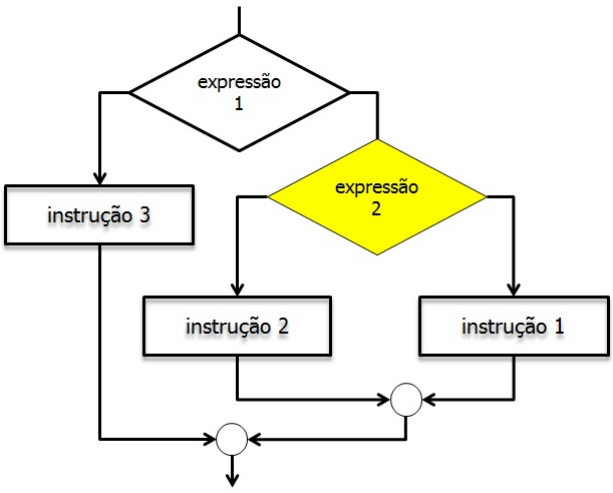

Figura 2: Estrutura de Seleção `if - if - else`.

onde a expressão booleana `frequencia >= 75` é avaliada, se ela for verdadeira será feita a avaliação `if` da expressão booleana `media >= 6.0` (linha 2), e se ela for verdadeira, a atribuição `'AP'` será feita e o programa será desviado para a primeira instrução após a linha 7. Se a expressão booleana `media >= 6.0` (linha 2) for falsa, o programa é desviado para a linha 4, (`else`) atribuindo `'RN'` a mensagem, e desviando o programa para a primeira instrução após a linha 7. Caso a expressão `frequencia >= 75` seja falsa, o programa será desviado para a linha 6 (`else` da mesma indentação do `if` da linha 1) e `msg = 'RF'` será atribuído a mensagem. Em qualquer caso, ao final de uma dessas atribuições, a execução do programa continua com a instrução seguinte ao final da indentação do bloco `if if else` (linha 7).

2 Resolvendo Problemas Numéricos com Estrutura de Seleção - 2

Como vimos anteriormente, na solução de vários vários problemas, só a estrutura de controle sequencial não é suficiente para projetar algoritmos (ou programas) que implementem soluções computacionais para a maior parte dos problemas. Além disso, para vários problemas, usando apenas uma comparação `if else` também não é suficiente. Vamos descrever vários exemplos de problemas que utilizam mais de uma comparação `if else` ou usam `if elif elif ... elif else` em suas soluções computacionais.

2.1 Lados de um triângulo

Um jardineiro deseja saber se dados três números reais eles podem ser os comprimentos dos lados de um triângulo. Caso os números formem um triângulo, o jardineiro deseja saber que tipo de triângulo ele pode formar com os lados (equilátero, isósceles ou escaleno). No caso de não formarem um triângulo, imprimir uma mensagem adequada.

Dados os lados 2.5, 3.0 e 2.5 para verificar se eles formam um triângulo, temos que analisar se todas as somas de duas das suas medidas são maiores que a terceira. Ou seja, para que os lados l_1, l_2 e l_3 formem um triângulo, temos que $l_1 + l_2 > l_3$, $l_1 + l_3 > l_2$ e $l_2 + l_3 > l_1$. Para o nosso exemplo, temos $2.5 + 3.0 > 2.5$, $2.5 + 2.5 > 3.0$ e $3.0 + 2.5 > 2.5$, logo temos que os lados formam um triângulo. Além disso, como dois dos lados são iguais

a 2.5, o triângulo é isósceles. Já no caso em que os lados são 1.0, 2.0 e 3.0 eles não formam um triângulo, pois a expressão $1.0 + 2.0 > 3.0$ não é verdadeira.

Exemplo 1

```
# formato de entrada
2.5
3.0
2.5

# formato de saída
lados: 2.50, 3.00, 2.50 - isósceles
```

Exemplo 2

```
# formato de entrada
1.0
2.0
3.0

# formato de saída
lados: 1.00, 2.00, 3.00 - não é triângulo
```

O problema pode ser descrito como:

```
# Este algoritmo lê três números reais e verifica se  eles podem ser os
# comprimentos dos lados de um  triângulo. Caso formem um triângulo,
# imprimir o tipo (equilátero, isósceles, escaleno) ou caso contrário,
# uma mensagem.
```

As especificações de entrada e saída são:

Entrada	Saída
3 lados. lado1, lado2, lado3 Os lados são números reais.	Uma mensagem informado que os lados dados formam um triângulo (equilátero, isósceles, escaleno) ou não formam um triângulo.

Como vimos acima, para que três números reais sejam os lados de um triângulo necessitamos que a soma de dois quaisquer dos lados seja sempre maior que o terceiro lado. Podemos expressar essa ideia usando a seguinte expressão:

```
(lado1 + lado2 > lado3) and (lado1 + lado3 > lado2) and
(lado2 + lado3 > lado1)
```

e as pré e pós-condições ficam:

```
# pré: lado1 lado2 lado3 and lado[i] > 0

# pós: (triangulo == (lado1 + lado2 > lado3) and (lado1 + lado3 > lado2)
#      and (lado2 + lado3 > lado1)) and (("Não é Triângulo" and
#      triangulo == false) or "equilátero" or "isósceles" or "escaleno")
```

Nesse problema as variáveis são compostas de tipos `float`, `bool` e `string`. Necessitamos do tipo `float` para armazenar os valores de `lado1`, `lado2` e `lado3` e da variável do tipo `bool` para armazenar o resultado da expressão `triângulo`. O tipo `string` será utilizado para armazenar a mensagem de classificação do triângulo ou de que os lados não formam um triângulo.

```
# descrição das variáveis utilizadas
# float lado1, lado2, lado3
# bool triangulo
# str msg
```

A seguir, descrevemos os passos do Algoritmo:

```
Algoritmo: Triângulo
# Passo 1. Leia os lados
# Passo 2. Verifique se formam um triângulo e classifique
# Passo 3. Imprima a classificação
# fim do algoritmo
```

O passo 2 não pode ser implementado diretamente num computador e necessitamos refinar esse esse passo.

```
# Passo 2. Verifique se formam um triângulo e classifique
# Passo 2.1. Atribua a variável booleana triangulo
# Passo 2.2. Se triângulo, classifique
# Passo 2.3. Caso contrário, atribua a mensagem correspondente
```

O Passo 2.2. ainda necessita de um refinamento. Temos que descrever como será a classificação dos triângulos Utilizaremos uma variável do tipo `string` para armazenar a classificação. O Passo 2.2. fica:

```
# Passo 2.2.1. Se os lados forem todos iguais classificação equilátero
# Passo 2.2.2. Se apenas dois lados forem iguais classificação isósceles
# Passo 2.2.3. Se nenhum dos lados forem iguais classificação escaleno
```

Vamos agora descrever a implementação em Python da solução desse problema. Os Passos 1 e 3, são de simples implementação. A implementação do Passo 2 precisa de mais um detalhamento. Vimos anteriormente que três números formam um triângulo se a soma de dois quaisquer sempre é maior que o terceiro. Considerando a variável booleana `triangulo`, o Passo 2.1. fica:

```
# Passo 2.1. Atribua a variável booleana triângulo
triangulo=(lado1+lado2>lado3) and (lado1+lado3>lado2) and
                                                  (lado2+lado3>lado1)
```

A variável `triangulo` só será verdeira se as três cláusulas forem verdadeiras. Caso contrário, o valor atribuído à variável será `false`.

Para implementar o Passo 2.2. em Python usamos as definições de triângulos equiláteros, isósceles e escaleno:

```
# Passo 2.2. Se triângulo, classifique
if triangulo:
# Passo 2.2.1. Se os lados forem todos iguais atribua equilátero
   if (lado1 == lado2) and (lado2 == lado3):
      classificacao = 'equilátero'
# Passo 2.2.2. Se apenas dois lados forem iguais isósceles
   elif (lado1==lado2) or (lado2==lado3) or (lado1==lado3):
      classificacao = 'isósceles'
# Passo 2.2.3. Se nenhum dos lados forem iguais escaleno
   else:
      classificacao = 'escaleno'
# Passo 2.3. Se não for triângulo atribua que não é triângulo
else:
   classificacao = 'não é triângulo'
```

A descrição completa do programa em Python fica:

```
# -*- coding: utf-8 -*-
# Programa: triangulo.py
# Programador:
# Data: 07/03/2020
# Este algoritmo lê três números reais e verifica se  eles podem ser os
# comprimentos dos lados de um  triângulo. Caso formem um triângulo,
# imprimir o tipo (equilátero, isósceles, escaleno) ou caso contrário,
# uma mensagem.
# início do módulo principal
# descrição das variáveis utilizadas
# float  lado1, lado2, lado3
# bool   triangulo
# string classificacao

# pré: lado1 lado2 lado3 && lado[i] > 0

# Passo 1. Leia os lados
print('Leia três numeros reais (lados): ')
lado1 = float(input())
lado2 = float(input())
lado3 = float(input())
# Passo 2. Verifique se formam um triângulo
# Passo 2.1. Atribua a variável booleana triangulo
triangulo=(lado1+lado2>lado3) and (lado1+lado3>lado2) and \
                                                  (lado2+lado3>lado1)
# Passo 2.2. Se triângulo, classifique
if triangulo:
# Passo 2.2.1. Se os lados forem todos iguais atribua equilátero
   if (lado1 == lado2) and (lado2 == lado3):
      classificacao = 'equilátero'
# Passo 2.2.2. Se apenas dois lados forem iguais isósceles
   elif (lado1==lado2) or (lado2==lado3) or (lado1==lado3):
```

```
33      classificacao = 'isósceles'
34  # Passo 2.2.3. Se nenhum dos lados forem iguais escaleno
35     else:
36        classificacao = 'escaleno'
37  # Passo 2.3. Se não for triângulo atribua que não é triângulo
38  else:
39     classificacao = 'não é triângulo'
40  # Passo 4. Imprima a classificação
41  print('lados: {0:.2f}, {1:.2f}, {2:.2f} - {3:s}'. \
42                                      format(lado1,lado2,lado3,msg))
43
44  # pós: (triangulo == (lado1 + lado2 > lado3) and (lado1 + lado3 > lado2)
45  #      and (lado2 + lado3 > lado1)) and (("Não é Triângulo" and
46  #      triangulo == false) or "equilátero" or "isósceles" or "escaleno")
47  # fim do módulo principal
```

2.2 Média Notas III

A avaliação da disciplina de Algoritmos e Estrutura de Dados I é dada por 3 notas de provas (`P1`, `P2` e `P3` e 2 notas dos trabalhos `T1` e `T2` . A média das provas `'MP'` é dada por `(P1 + P2 + P3)/3.0` e a média dos trabalhos `'MT'` é dada por `(T1 + T2)/2.0`. A média da disciplina `'MF'` é dada pela média ponderada `0.75*MP + 0.25*MT` das avaliações. A média é calculada com uma casa decimal. Valores maiores ou iguais a cinco na segunda casa decimal são aproximados para um décimo. Valores menores que cinco na segunda casa decimal não são considerados. O(a)s estudantes com média maiores ou iguais a 6.0 são considerados aprovados e os alunos com média inferior a 6.0 não. O programa `medianotas02.py`, descrito anteriormente, apresenta uma solução para esse problema. Considerando que todo(a)s o(a)s estudantes podem fazer uma prova optativa, e que a nota dessa prova substitui a menor nota das avaliações das provas, queremos agora estender a solução `medianotas02.py` para resolver o cálculo da média levando em consideração a prova optativa, e em função disso, temos que reprogramar o cálculo da média final. O programa deve ter uma saída de acordo com as especificações dos exemplos abaixo, imprimindo a mensagem correspondente, `'AP'` ou `'RP'` para cada estudante.

Vamos descrever as alterações necessárias para incorporar a prova optativa no cálculo da média da disciplina. A entrada do algoritmo é o RGA do(a) estudante, as notas das três provas, e a nota dos dois trabalhos. Além disso, o algoritmo deve ler se o(a) estudante fez a prova optativa, e nesse caso deve ler a nota da prova optativa.

Exemplo - com prova optativa

```
# formato da entrada
20181910045-6 # entrada do número do estudante
6.0 7.0 2.0 # entrada das notas das provas
9.0 9.0 # entrada das notas dos trabalhos
S # resposta se o aluno fez a prova optativa
6.0 # nota da prova optativa

# formato da saída
RGA: 20181910045-6: Média =  7.0 - AP
```

Exemplo - sem prova optativa

```
# formato da entrada
20181910048-6 # entrada do número do estudante
6.0 7.0 6.0 # entrada das notas das provas
9.0 9.0 # entrada das notas dos trabalhos
N # resposta se o aluno fez a prova optativa

# formato da saída
RGA: 20181910048-6: Média =  7.0 - AP
```

No caso da descrição do problema, basta acrescentarmos a possibilidade do estudante fazer a prova optativa e se a nota da prova optativa for maior que a menor nota das provas, deve-se substituir a menor nota pela nota da prova optativa.

```
# Este programa calcula a média final de um aluno para um curso com
# o total de três notas de provas e duas notas de trabalhos. A média
# da disciplina é dada por 0.75 da média das provas somada com 0.25
# da média dos trabalhos. O alunos pode fazer uma prova optativa que
# substitui a menor nota das provas. Caso a média do aluno seja maior
# ou igual a 6.0 o aluno é considerado aprovado ('AP'), caso contrário,
# o aluno é considerado reprovado ('RP').
```

Além das variáveis do programa `medianotas02.py`, necessitamos de variáveis para ler a resposta `resp` do usuário se o estudante fez a prova optativa e a respectiva nota `opt`. A variáveis são do tipo `string` e `float`, respectivamente. Além disso, também precisaremos de uma variável do tipo `float` para armazenar a menor nota que poderá ser substituída pela nota da prova optativa.

```
# Descrição das variáveis utilizadas
# float prova1, prova2, prova3, trab1, trab2
# float mediaprovas, mediatrab, mediafinal
# float opt, menornota
# str rga, msg, resp
```

e as pré e pós-condições ficam:

```
# pré: prova1, prova2, prova3, trab1, trab2

# pós: mediafinal and ('AP' and mediafinal >= 6.0 or 'RP') and
#      mediafinal = 0.75*mediaprovas + 0.25*mediatrab and
#      (mediaprovas == (prova1 + prova2 + prova3) / 3.0 or
#      0.75*(prova1+prova2+prova3-menornota+opt)/3.0 and
#      menornota == min(prova1,prova2,prova3)) and
#      mediatrabalhos == (trab1 + trab2) / 2.0)
```

Alterando o Passo 1. do algoritmo anterior, e usando as novas variáveis, temos:

```
# Passo 1. Leia o numero e as notas das provas e trabalhos
# Passo 1.1. Leia o RGA do aluno
rga = input('Leia o número do estudante: ')
# Passo 1.2. Leia as notas das provas
print('Entre com a nota das provas:')
prova1, prova2, prova3 = map(float,input().split())
# Passo 1.3. Leia a nota dos trabalhos.
print('Entre com a nota dos trabalhos:')
trab1, trab2 = map(float,input().split())
# Passo 1.4. Verifique se o(a) estudante fez a prova optativa
print('O Aluno fez prova optativa (S/N)?:')
resp = input()
if resp.upper() == 'S':
# Passo 1.4.1. Se fez prova optativa então leia a nota
   print('Entre com a nota da prova optativa:')
   opt = float(input())
```

onde a instrução `resp.upper()` garante que todo caractere que for digitado será transformado em maiúsculo e com isso simplifica a comparação.

Após a leitura das notas, temos que computar a média final do estudante. Como o estudante pode ter feito ou não a prova optativa, temos que projetar uma solução que atenda essas duas possibilidades. No caso em que o estudante não fez a prova optativa, a solução é a mesma do programa `medianotas02.py`. Se o estudante fez a prova optativa, para calcular a média final temos que computar qual a menor nota das provas e verificar se a nota da prova optativa é maior que a menor nota das provas. Para computar qual é a menor nota das provas, podemos assumir que a menor nota é a nota da primeira prova, depois comparamos a menor nota com a nota da segunda prova, se a nota da segunda prova for menor, atribuímos para a menor nota a nota da segunda prova e finalmente, comparamos a menor nota com a nota da terceira prova, se a nota da terceira prova for menor, atribuímos para a menor nota a nota da terceira prova.

```
# Passo 2.1. Calcule a média das provas usando a prova optativa
if resp.upper() == 'S':
# Passo 2.1.1. Calcule a menor nota
# Passo 2.1.1.1. Atribua prova1 a menornota
   menornota = prova1
# Passo 2.1.1.2. Se prova2 < menornota atribua prova2 a menornota
   if prova2 < menornota:
      menornota = prova2
# Passo 2.1.1.3. Se prova3 < menornota atribua prova3 a menornota
   if prova3 < menornota:
      menornota = prova3
# Passo 2.2.2. Se opt > menornota calcule a média com a opt
   if opt > menornota:
      mediaprovas = (prova1 + prova2 + prova3 - menornota + opt)/3.0
# Passo 2.2.3. Caso contrário calcule a média sem a opt
   else:
      mediaprovas = (prova1 + prova2 + prova3)/3.0
```

No caso do cálculo da média, se o estudante não fez a prova optativa ou a nota da prova optativa for menor que as notas de todas as provas, o cálculo da média das provas só levará em consideração as notas das três provas e nesse caso, o cálculo da média não se altera:

```
mediaprovas = (prova1 + prova2 + prova3)/3.0
```

Se o estudante fez a prova optativa `opt` e a nota é maior que a menor nota das provas (`menornota = min(nota1, nota2, nota3)`), no cálculo da média das provas temos que substituir o valor da menor nota pelo valor da nota da prova optativa. Apesar se conhecermos o valor da menor nota, não sabemos se ela é o valor da `nota1`, `nota2` ou `nota3`. Uma forma de obter a média das provas sem a necessidade de saber qual é a menor nota é somarmos todas as notas e subtrairmos o valor da menor nota, como isso, temos certeza que o valor da soma só leva em consideração as duas maiores notas. Com isso, basta somarmos o valor da prova optativa que ficaremos com a soma das três maiores notas entre `prova1`, `prova2`, `prova3` e `opt`.

```
mediaprovas = (prova1 + prova2 + prova3 - menornota + opt)/3.0
```

No exemplo a seguir, ilustramos o caso da entrada em que o estudante fez a prova optativa:

Exemplo

```
# formato da entrada
20181910045-6 # entrada do número do estudante
6.0 7.0 2.0 # entrada das notas das provas
9.0 9.0 # entrada das notas dos trabalhos
S # resposta se o aluno fez a prova optativa
6.0 # nota da prova optativa

# estado das variáveis após a leitura
rga == '20181910045-6'
nota1 == 6.0
nota2 == 7.0
nota3 == 2.0
trab1 == 9.0
trab2 == 9.0
resp == 'S'
opt == 6.0

# cálculo da média das provas
if resp.upper() == 'S':
if 'S' == 'S': # True
   menornota = prova1
   menornota = 6.0
   if prova2 < menornota:
   if 7.0 < 6.0: # False
   if prova3 < menornota:
   if 2.0 < 6.0: # True
      menornota = 2.0
   if opt > menornota:
```

```
   if 6.0 > 2.0: # True
      mediaprovas = (prova1 + prova2 + prova3 - menornota + opt)/3.0
      mediaprovas = (6.0 + 7.0 + 2.0 - 2.0 + 6.0)/3.0
      mediaprovas = 6.333333

# o fluxo do programa é desviado e a instrução
# mediaprovas = (prova1+prova2+prova3)/3.0 não é executada

# estado das variáveis após o cálculo da média das provas
rga == '20181910045-6'
nota1 == 6.0
nota2 == 7.0
nota3 == 2.0
trab1 == 9.0
trab2 == 9.0
resp == 'S'
opt == 6.0
menornota == 2.0
mediaprovas == 6.333333
```

Os demais passos seguem a descrição do programa `medianotas02.py`. A descrição completa em Python do programa Média Notas com prova optativa fica:

```
1  # -*- coding: utf-8 -*-
2  # Programa: medianotas03.py
3  # Programador:
4  # Data: 25/11/2016
5  # Este programa calcula a média final de um aluno para um curso com
6  # o total de três notas de provas e duas notas de trabalhos. A média
7  # da disciplina é dada por 0.75 da média das provas somada com 0.25
8  # da média dos trabalhos. O alunos pode fazer uma prova optativa que
9  # substitui a menor nota das provas. Caso a média do aluno seja maior
10 # ou igual a 6.0 o aluno é considerado aprovado ('AP'), caso contrário,
11 # o aluno é considerado reprovado ('RP').
12 # início do módulo principal
13 # descrição das Variáveis utilizados
14 # float prova1, prova2, prova3, trab1, trab2
15 # float mediaprovas, mediatrab, mediafinal
16 # float opt, menornota
17 # str rga, msg, resp
18
19 # pré: rga prova1 prova2 prova3 trab1 trab2 and
20 #      (('S' opt) or 'N')
21
22 # Passo 1. Leia o numero e as notas das provas e trabalhos
23 # Passo 1.1. Leia o RGA do aluno
24 rga = input('Leia o número do estudante: ')
25 # Passo 1.2. Leia as notas das provas
26 print('Entre com a nota das provas:')
```

```
prova1, prova2, prova3 = map(float,input().split())
# Passo 1.3. Leia a nota dos trabalhos.
print('Entre com a nota dos trabalhos:')
trab1, trab2 = map(float,input().split())
# Passo 1.4. Verifique se o(a) estudante fez a prova optativa
print('O Aluno fez prova optativa ('S'/'N')?:')
resp = input()
if resp.upper() == 'S':
# Passo 1.4.1. Se fez prova optativa então leia a nota
   print('Entre com a nota da prova optativa:')
   opt = float(input())
# Passo 2. Calcule a média da disciplina e a mensagem
# Passo 2.1. Calcule a média das provas usando a prova optativa
if resp.upper() == 'S':
# Passo 2.1.1. Calcule a menor nota
# Passo 2.1.1.1. Atribua prova1 a menornota
   menornota = prova1
# Passo 2.1.1.2. Se prova2 < menornota atribua prova2 a menornota
   if prova2 < menornota:
      menornota = prova2
# Passo 2.1.1.3. Se prova3 < menornota atribua prova3 a menornota
   if prova3 < menornota:
      menornota = prova3
# Passo 2.2.2. Se opt > menornota calcule a média com a opt
   if opt > menornota:
      mediaprovas = (prova1 + prova2 + prova3 - menornota + opt)/3.0
# Passo 2.2.3. Caso contrário calcule a média sem a opt
   else:
      mediaprovas = (prova1 + prova2 + prova3)/3.0
# Passo 2.2. Calcule a média das notas sem prova optativa
else:
   mediaprovas = (prova1 + prova2 + prova3)/3.0
# Passo 2.3. Calcule a média dos trabalhos
mediatrab = (trab1 + trab2)/2.0
# Passo 2.4. Calcule a média final da disciplina
mediafinal = 0.75 * mediaprovas + 0.25 * mediatrab
# Passo 2.5. Arredonde a media final com uma casa decimal
mediafinal = round(mediafinal,1)
# Passo 2.6. Compute a mensagem correspondente
if mediafinal >= 6.0:
   msg = 'AP'
else:
   msg = 'RP'
# Passo 3. Imprima os resultados
print('RGA: {0:s}: Média ={1:5.1f} - {2:s}'.format(rga,mediafinal,msg))

# pós: (('AP' and Final >=6.0) or 'RN') and mediaFinal == 0.75*mediaProvas
#      + 0.25*mediaTrabalhos and (mediaBotas == (prova1 + prova2 + prova3
#      + optativa - menorNota)/3.0 and resposta == 'S' and optativa >
#      menorNota) or mediaNotas == (prova1 + prova2 + prova3)/3.0 and
```

```
77 #      mediaTrabalhos == (trabalho1 + trabalho2)/2.0 or (mediaAvaliacoes
78 #      == (prova1 + prova2 + prova3 + trabalho - menorNota + optativa/4.0)
79 #      and menoNota == min{prova1, prova2, prova3}
80 # fim do módulo principal
```

Implemente o programa, interprete e execute para as entradas abaixo. Verifique se a saída do seu programa está correta.

Exemplo 1

```
# formato da entrada
20181910045-6
6.0 7.0 6.0
9.0 9.0
N

# formato da saída
RGA: 20181910045-6: Média =  7.0 - AP
```

Exemplo 2

```
20181910046-3
5.0 3.0 5.0
6.0 6.0
S
8.0

# formato da saída
20181910046-3: Média =  6.0 - AP
```

2.3 Poluição do Ar II

Anteriormente, analisamos o Problema da Poluição do Ar, onde após o cálculo do índice de poluição, eram atribuídas duas mensagens, uma para valores do índice menores que um valor limite e outra para valores maiores ou iguais ao valor limite. Vamos agora ampliar a solução do Problema da Poluição do Ar para o caso onde a decisão abrange mais de duas possibilidades. Considere o mesmo problema de computar o nível de poluição do ar na cidade de Cachorro Sentado, onde o índice de poluição é medido pela média de três leituras que são feitas às 12:00 hs na Olaria da Sucuri, no Bolicho do Jaburu, e num ponto aleatório na vila das Emas. O índice de poluição, um número de ponto flutuante, é dado pela média aritmética dessas três leituras de números inteiros. Valores do índice de poluição abaixo ou igual a 50 indicam **qualidade do ar boa**, valores menores ou iguais a 100 indicam **qualidade do ar regular**, valores menores que 200 indicam **qualidade do ar inadequada**, valores menores que 300 indicam **qualidade do ar má**, enquanto valores maiores ou iguais a 300 indicam **qualidade do ar péssima**. Projete e implemente um programa que leia as medidas de poluição, calcule o índice de poluição e então determine e imprima a mensagem com a condição a condição apropriada: `Qualidade do Ar Boa`, `Qualidade do Ar Regular`, `Qualidade do Ar Inadequada`, `Qualidade do Ar Má` e `Qualidade do Ar Péssima`.

Exemplo 1

```
# formato da entrada
65
35
33

# formato da saída
Qualidade do Ar Boa
```

Exemplo 2

```
# formato da entrada
38 72 71

# formato da saída
Qualidade do Ar Regular
```

Na descrição do problema incorporamos os diferentes níveis de poluição:

```
# Este programa lê três níveis de poluição do ar, leitura1, leitura2 e
# leitura3. A média aritmética dessas três leituras é usada como o índice
# de poluição da cidade, valores desse  índice de poluição do ar abaixo ou
# iguais a 50 indicam Qualidade do Ar Boa, valores menores ou iguais a 100
# indicam Qualidade do Ar Regular, valores menores que 200 indicam
# Qualidade do Ar Inadequada, valores menores que 300 indicam Qualidade do
# Ar Má e maiores ou iguais a 300 indicam Qualidade do Ar Péssima. O
# programa calcula o índice de poluição do ar e então determina e imprime
# a condição apropriada do ar, Boa, Regular, Inadequada Má ou Péssima.
```

As especificações de entrada e saídaficam:

Entrada	**Saída**
3 números inteiros representado as leituras de umidade relativa do ar	índice de poluição e msg “Qualidade do ar Boa” ou “Qualidade do Ar Regular” ou “Qualidade do Ar Inadequada” ou “Qualidade do Ar Má” ou “Qualidade do Ar Péssima”.

Para os identificadores, não necessitamos da constante `LIMITE` e as variáveis e seus respectivos tipos para armazenar as informações de entrada e saída são são compostas de tipos básicos. A linguagem Pyhon usa tipagem dinâmica, independente disso, é importante descrever as variáveis que serão utilizadas no programa. No nosso caso, necessitaremos de variáveis do tipo `int` para armazenar os valores de `leitura1`, `leitura2` e `leitura3` e tipo `float` para armazenar o `indice`. Além disso, necessitamos de uma variável do tipo `string`, `msg` para armazenar a mensagem correspondente. Quando da implementação do algoritmo pode surgir a necessidade da utilizarmos variáveis auxiliares para armazenar valores intermediários.

```
# descrição das variáveis utilizadas
# int leitura1, leitura2, leitura3
# float indice
# str msg
```

sendo que as pré e pós-condições ficam:

```
# pré: leitura1 leitura2 leitura3

# pós: indice == (Soma i em {1,2,3}: leitura[i])/3.0 and
#      (indice <= 50 and 'Qualidade Boa') or (Indice <= 100  and
#      'Qualidade Regular') or (Indice < 200 and 'Qualidade Inadequada')
#      or (Indice  < 300 and 'Qualidade Má') or 'Qualidade Péssima'
```

Exemplo 1

```
# formato da entrada
Entre com a primeira medida de poluição: 65
Entre com a segunda medida de poluição: 35
Entre com a terceira medida de poluição: 31

# estado das variáveis após a leitura dos dados da entrada
leitura1 == 65
leitura2 == 35
leitura3 == 33

# cálculo do índice
indice = (leitura1 + leitura2 + leitura3) / 3.0
indice == 44.333333

# formato da saída
Qualidade do Ar Boa
```

Exemplo 2

```
# formato da entrada
Entre com a primeira medida de poluição: 38
Entre com a segunda medida de poluição: 72
Entre com a terceira medida de poluição: 71

# estado das variáveis após a leitura dos dados da entrada
leitura1 == 38
leitura2 == 72
leitura3 == 71

# cálculo do índice
indice = (leitura1 + leitura2 + leitura3) / 3.0
indice == 60.333333
```

```
# formato da saída
Qualidade do Ar Regular
```

Como descrevemos acima, a principal diferença deste problema com o problema apresentado anteriormente é a classificação da qualidade do ar. No problema anterior tínhamos apenas duas possibilidades, agora temos cinco classificações. Em função disso, o Passo 3 do algoritmo necessita um refinamento, detalhar os sub-passos de forma que eles possam ser implementados num computador.

```
# Passo 3. Compute a mensagem relativa ao índice de poluição
# Passo 3.1. Se o índice for <= 50 qualidade boa
# Passo 3.2. Senão se índice <= 100 qualidade regular
# Passo 3.3. Senão se índice < 200 qualidade inadequada
# Passo 3.4. Senão se índice < 300 qualidade má
# Passo 3.5. Senão qualidade péssima
```

Essas instruções condicionais podem sem implementadas em Python com a instrução `if elif else`:

```
# Passo 3. Compute a mensagem relativa ao índice de poluição
# Passo 3.1. Se o índice for <= 50 qualidade boa
if indice <= 50:
   msg = 'Qualidade do Ar Boa'
# Passo 3.2. Senão se índice <= 100 qualidade regular
elif indice <= 100:
   msg = 'Qualidade do Ar Regular'
# Passo 3.3. Senão se índice < 200 qualidade inadequada
elif indice < 200:
   msg = 'Qualidade do Ar Inadequada'
# Passo 3.4. Senão se índice < 300 qualidade má
elif indice < 300:
   msg = 'Qualidade do Ar Má'
# Passo 3.5. Senão qualidade péssima
else:
   msg = 'Qualidade do Ar Péssima'
```

Com isso nosso programa em Python fica:

```
# -*- coding: utf-8 -*-
# Programa: poluicao2.py
# Programador:
# Data: 22/09/2016
# Este programa lê três níveis de poluição do ar, leitura1, leitura2 e
# leitura3. A média aritmética dessas três leituras é usada como o índice
# de poluição da cidade, valores desse  índice de poluição do ar abaixo ou
# iguais a 50 indicam Qualidade do Ar Boa, valores menores ou iguais a 100
# indicam Qualidade do Ar Regular, valores menores que 200 indicam
# Qualidade do Ar Inadequada, valores menores que 300 indicam Qualidade do
```

```
# Ar Má e maiores ou iguais a 300 indicam Qualidade do Ar Péssima. O
# programa calcula o índice de poluição do ar e então determina e imprime
# a condição apropriada do ar, Boa, Regular, Inadequada Má ou Péssima.
# início do módulo principal
# descrição das variáveis utilizados
# int leitura1, leitura2, leitura3
# float indice
# str msg

# pré: leitura1 leitura2 leitura3

# Passo 1. Leia as medidas de poluição
leitura1 = int(input('Entre com a primeira medida de poluição: '))
leitura2 = int(input('Entre com a segunda medida de poluição: '))
leitura3 = int(input('Entre com a terceira leitura de poluição: '))
# Passo 2. Calcule o índice de poluição
índice = (leitura1 + leitura2 + leitura3)/3.0
# Passo 3. Compute a mensagem relativa ao índice de poluição
# Passo 3.1. Se o índice for <= 50 qualidade boa
if indice <= 50:
   msg = 'Qualidade do Ar Boa'
# Passo 3.2. Senão se índice <= 100 qualidade regular
elif indice <= 100:
   msg = 'Qualidade do Ar Regular'
# Passo 3.3. Senão se índice < 200 qualidade inadequada
elif indice < 200:
   msg = 'Qualidade do Ar Inadequada'
# Passo 3.4. Senão se índice < 300 qualidade má
elif indice < 300:
   msg = 'Qualidade do Ar Má'
# Passo 3.5. Senão qualidade péssima
else:
   msg = 'Qualidade do Ar Péssima'
# Passo 4. Imprima a mensagem
print('{0:s}'.format(mensagem))

# pós: indice == (Soma i em {1,2,3}: lista[i])/3.0 and
#      (indice <= 50 and 'Qualidade Boa') or (Indice <= 100  and
#      "Qualidade Regular") or (Indice < 200 and 'Qualidade Inadequada')
#      or (Indice  < 300 and 'Qualidade Má') or 'Qualidade Péssima'
# fim do módulo principal
```

3 Resolvendo Problemas de Texto com Estrutura de Seleção

3.1 Codificando Palavras

Como observamos anteriormente, a desvantagem do cifrador de César é que ele é relativamente fácil de decodificar. Como apresentado anteriormente, uma estratégia de codificação

que é mais difícil de decodificar usa uma permutação aleatória do alfabeto e substitui os caracteres da palavra a ser codificada usando essa permutação. Considere a permutação do alfabeto abaixo:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
JSATERNICFHPLUGWVYDZOMXQBK
```

ela codifica palavras com 5 caracteres como:

Exemplo 1

```
# formato de entrada
Maria # palavra de entrada

# formato da saída
LJYCJ
```

Exemplo 2

```
# formato de entrada
AULAS

# formato da saída
JOPJD
```

Exemplo 3

```
# formato de entrada
123!?

# formato da saída
123!?
```

Mesmo com o uso de permutações, se a mensagem for muito grande, a mensagem pode ser decodificada estimando a frequência das letras no texto em função do idioma utilizado. Existem formas de dificultar a decodificação com o uso de diferentes permutações para blocos de caracteres.

Como no exemplo anterior, a descrição detalhada para esse problema é praticamente a própria descrição do problema.

```
# Este algoritmo lê usa uma permutação do alfabeto e lê uma palavra de
# cinco caracteres e usando a permutação codifica a palavra e imprime a
# palavra codificada.
```

As especificações de entrada e saída ficam:

Entrada	Saída
uma permutação do alfabeto `'A'..'Z'` uma palavra com cinco caracteres.	uma palavra codificada

Nesse exemplo as variáveis e constantes são compostas de tipos básicos. Necessitamos uma constante do tipo `string` para armazenar uma permutação do alfabeto `PERMUTACAO` e variáveis do tipo `string` para armazenar a palavra com 5 caracteres `palavra` e a respectiva codificação `palavraC`.

```
# descrição das constantes e variáveis utilizadas
# string PERMUTACAO
# string palavra, palavraC
# string char0, char1. char2, char3, char4
```

sendo que as pré e pós-condições ficam:

```
# pré: PERMUTACAO palavra

# pós: palavraC[0]..palavraC[4] and para i em {0,..,4}:
#      palavraC[i] == PERMUTACAO(ord(palavra[i]-65)
```

Os passos do Algoritmo são os mesmos do exemplo anterior:

```
Algoritmo: CodifiquePalavra
# Passo 1. Inicialize a permutação e leia a palavra
# Passo 2. Codifique os caracteres da palavra
# Passo 3. Imprima a palavra codificada
# fim do algoritmo
```

Temos que detalhar com o Passo 2 será executado. Necessitamos descrever como a codificação da palavra será feita. Para cada caractere da palavra, associamos um índice de `0,..,25` e selecionamos o caractere correspondente ao índice na `PERMUTACAO`. Para obter o índice de um caractere usamos a função `ord()`, que retorna o código ASCII do caractere. Como os códigos ASCII dos caracteres maiúsculos do alfabeto `'A',..,'Z'` variam de 85 a 90, temos que subtrair 85 para ficarmos no intervalo $[0, 25]$.

Exemplo

```
# formato de entrada
PERMUTACAO = 'JSATERNICFHPLUGWVYDZOMXQBK'
Maria # palavra de entrada

# estado das variáveis e constantes após a leitura dos dados
PERMUTACAO == 'JSATERNICFHPLUGWVYDZOMXQBK'
palavra == 'Maria'

# Codificação dos caracteres
# verifique se o caractere palavra[0] é do alfabeto
if palavra[0].isalpha():
if 'M'.isalpha(): # True
# transforme o caractere para maiúsculo
   char0 = palavra[0].upper()
   char0 = 'M'

# codifique o caractere char0
```

```
   char0 = PERMUTACAO[ord(char0) - ord('A')]
   char0 = PERMURACAO[ord('M') - 65]
   char0 = PERMUTACAO[77-65]
   char0 = PERMUTACAO[12]
   char0 = 'L'

# verifique se o caractere palavra[1] é do alfabeto
if palavra[1].isalpha():
if 'a'.isalpha(): # True
# transforme o caractere para maiúsculo
   char1 = palavra[1].upper()
   char1 == 'A'

# codifique o caractere char1
   char1 = PERMUTACAO[ord(char0) - ord('A')]
   char1 = PERMURACAO[ord('A') - 65]
   char1 = PERMUTACAO[65-65]
   char1 = PERMUTACAO[0]
   char1 = 'J'

# a codificação é feita para os demais caracteres char2, char3. char4

# formato da saída
LJYCJ
```

Uma implementação em Python é dada por:

```
# -*- coding: utf-8 -*-
# Programa: codifica30.py
# Programador:
# Data: 03/06/2016
# Este algoritmo lê usa uma permutação do alfabeto e lê uma palavra de
# cinco caracteres e usando a permutação codifica a palavra e imprime a
# palavra codificada.
# início do módulo principal
# descrição das variáveis e constantes utilizadas
# string PERMUTACAO
# string palavra, palavraC
# string char0, char1. char2, char3, char4

# pré: PERMUTACAO palavra

# Passo 1. Inicialize a permutação e leia a palavra
# Passo 1.1. Inicialize a permutação
PERMUTACAO = 'JSATERNICFHPLUGWVYDZOMXQBK'
# Passo 1.2. Leia a palavra a ser codificada
palavra = input()
# Passo 2. Codifique os caracteres da palavra
# Passo 2.1. Codifique o primeiro caractere
```

```
# Passo 2.1.1. Se o caractere for do alfabeto
if palavra[0].isalpha():
# Passo 2.1.1.1. Transforme o caractere para maiúsculo
   char0 = palavra[0].upper()
# Passo 2.1.1.2. Codifique o caractere
   char0 = PERMUTACAO[ord(char0) - ord('A')]
# Passo 2.1.2. Caso contrário
else:
# Passo 2.1.2.1. Mantenha o caractere
   char0 = palavra[0]
# Passo 2.2. Codifique o segundo caractere
# Passo 2.2.1. Se o caractere for do alfabeto
if palavra[1].isalpha():
# Passo 2.2.1.1. Transforme o caractere para maiúsculo
   char1 = palavra[1].upper()
# Passo 2.2.1.2. Codifique o caractere
   char1 = PERMUTACAO[ord(char1) - ord('A')]
# Passo 2.2.2. Caso contrário
else:
# Passo 2.2.2.1. Mantenha o caractere
   char1 = palavra[1]
# Passo 2.3. Codifique o terceiro caractere
# Passo 2.3.1. Se o caractere for do alfabeto
if palavra[2].isalpha():
# Passo 2.3.1.1. Transforme o caractere para maiúsculo
   char2 = palavra[2].upper()
# Passo 2.3.1.2. Codifique o caractere
   char2 = PERMUTACAO[ord(char2) - ord('A')]
# Passo 2.3.2. Caso contrário
else:
# Passo 2.3.2.1. Mantenha o caractere
   char2 = palavra[2]
# Passo 2.4. Codifique o quarto caractere
# Passo 2.4.1. Se o caractere for do alfabeto
if palavra[3].isalpha():
# Passo 2.4.1.1. Transforme o caractere para maiúsculo
   char3 = palavra[3].upper()
# Passo 2.4.1.2. Codifique o caractere
   char3 = PERMUTACAO[ord(char3) - ord('A')]
# Passo 2.4.2. Caso contrário
else:
# Passo 2.4.2.1. Mantenha o caractere
   char3 = palavra[3]
# Passo 2.5. Codifique o quinto caractere
# Passo 2.5.1. Se o caractere for do alfabeto
if palavra[4].isalpha():
# Passo 2.5.1.1. Transforme o caractere para maiúsculo
   char4 = palavra[4].upper()
# Passo 2.5.1.2. Codifique o caractere
   char4 = PERMUTACAO[ord(char4) - ord('A')]
```

```
# Passo 2.5.2. Caso contrário
else:
# Passo 2.5.2.1. Mantenha o caractere
   char4 = palavra[4]
# Passo 2.6. Concatene os caracteres
palavraC = char0 + char1 + char2 + char3+ char4
# Passo 3. Imprima a mensagem codificada
print('Codificada: {0:s}'.format(palavraC))

# pós: palavraC[0]..palavraC[4] and para i em {0..4}:
#      palavraC[i] == PERMUTACAO(ord(palavra[i]-65)
# fim do módulo principal
```

4 Resolvendo Problemas Gráficos com Estrutura de Seleção

4.1 Tiro ao Alvo

Nas aulas anteriores usamos a biblioteca `stddraw.py` para imprimir figuras geométricas num janela gráfica. Também vimos que podemos usar o mouse para passar informações para o programa. Na aula de hoje vamos simular um jogo de tiro ao alvo. Além da utilização do mouse, vamos incorporar a utilização de sons para animar as tentativas. A biblioteca `stdaudio.py` possui funções (métodos) para usar sons num programa. Vamos usar também a função da biblioteca `stddraw` para verificar se o mouse foi pressionado sobre a janela gráfica e "ler" as coordenadas (x, y) onde o mouse foi pressionado.

Considere o problema de pressionar o mouse dentro da área definida por um círculo e marcar as o ponto onde o mouse foi pressionado. Além disso, ao pressionar o mouse, gerar o som de um laser. Na Aula 08 usamos o mouse para definir os movimentos no jogo da velha, agora vamos usar o mouse para simular um tiro num alvo e marcar o tiro com um ponto vermelho no alvo, e "tocar" um som de laser a cada tiro.

Uma implementação em Python para esse jogo é dado por:

```
# -*- coding: utf8 -*-
# Programa: tiroaoalvo00.py
# Programador:
# Data: 03/03/2020
# Este programa utiliza as classes stddraw e stdio. Ele gera e
# imprime um círculo, lê o movimento do mouse e ao clique do
# mouse dispara um dardo na região gráfica e marca a posição# do clique. Além disse ge
# pressionado.
# Declaração das classes utilizadas
import stddraw
import stdaudio
# início do módulo principal
# descrição das variáveis e constantes utilizadas
# float x1, y1, x2, y2

# pré: x1, y1, x2, y2, dimensões e posição de um círculo na janela
#      gráfica.
```

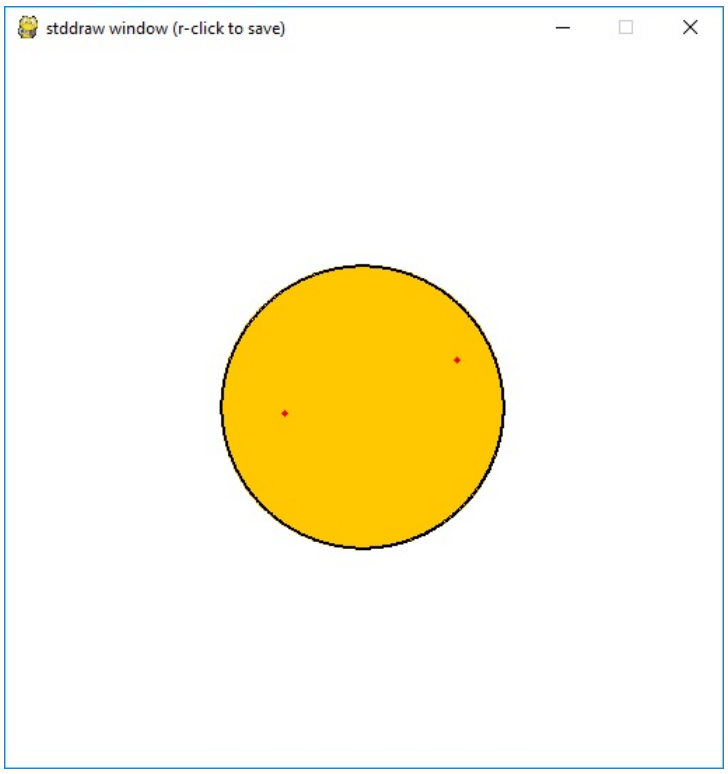

Figura 3: Tiro ao Alvo.

```
18
19 # Passo 1. Gere o círculo
20 # Passo 1.1. Defina a cor laranja do fundo do painel
21 stddraw.setPenColor(stddraw.ORANGE)
22 # Passo 1.2. Gere um círculo de fundo laranja
23 stddraw.filledCircle(0.5, 0.5, 0.2)
24 # Passo 1.3. Defina a cor para gerar as bordas do círculo
25 stddraw.setPenColor(stddraw.BLACK)
26 # Passo 1.4. Gere as bordas do círculo
27 stddraw.circle(0.5, 0.5, 0.2)
28 # Passo 2. Imprima o gráfico na tela
29 stddraw.show(1000.0)
30 # Passo 3. Leia os movimentos do mouse
31 # Passo 3.1. Defina a cor para o movimento do mouse
32 stddraw.setPenColor(stddraw.RED)
33 # Passo 3.2. Verifique se o mouse foi pressionado
34 if stddraw.mousePressed():
35 # Passo 3.2.1. Leia as coordenadas onde o mouse foi pressionado
36    x1 = stddraw.mouseX()
37    y1 = stddraw.mouseY()
38 # Passo 3.2.2. Verifique se as coordenadas estão no círculo
39 if (x1-.5)**2 + (y1-.5)**2 <= .2*.2:
40 # Passo 3.2.2.1. Marque as coordenadas (x1,y1)
41    stddraw.point(x1,y1)
42 # Passo 3.2.2.2. Gere o som
43    stdaudio.playFile('laser')
44 # Passo 3.2.2.2. Imprima as informações na tela
```

```
45      stddraw.show(1000.0)
46  # Passo 3.3. Verifique se o mouse foi pressionado
47  if stddraw.mousePressed():
48  # Passo 3.3.1. Leia as coordenadas onde o mouse foi pressionado
49      x2 = stddraw.mouseX()
50      y2 = stddraw.mouseY()
51  # Passo 3.3.2. Verifique se as coordenadas estão no círculo
52  if (x2-.5)**2 + (y2-.5)**2 <= .2*.2:
53  # Passo 3.3.2.1. Marque as coordenadas (x2,y2)
54      stddraw.point(x2,y2)
55  # Passo 3.3.2.2. Gere o som
56      stdaudio.playFile('laser')
57  # Passo 3.3.2.2. Imprima as informações na tela
58      stddraw.show(1000.0)
59
60  # pós: Um alvo com dois pontos marcados.
61  # fim do módulo principal
```

Este exemplo descreve instruções de como ler os movimentos do mouse numa janela gráfica. Além disso, detalhamos um pouco mais a função `show()`, pois precisamos que a a janela gráfica esteja ativa para que possamos ler os movimentos do mouse. Posteriormente utilizaremos a biblioteca `pygame` para implementar esse problema.

```
# Passo 2. Imprima o tabuleiro do jogo na tela
stddraw.show(1000.0)
```

No exemplo da aula de hoje, também introduzimos o conceito de como reproduzir um som num programa Python. Para isso, usamos a função `playFile()` da biblioteca `stdaudio.py`.

```
# Passo 3.3.2.2. Gere o som
    stdaudio.playFile('laser')
```

Para que o som de um laser seja reproduzido durante a execução do programa, o arquivo `lase.wav` deve estar no mesmo diretório em que o programa estiver sendo executado.

Edite, implemente e execute o programa `tiroaoalvo00.py`. Use a sua criatividade para mudar as cores e melhorar a apresentação do alvo.