Algoritmos e Programação I: Aula 05. *

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

Sumário

1 Problemas Computacionais Numéricos

Nas aulas 2 a 4, descrevemos vários exemplos de algoritmos que solucionam problemas usando uma estrutura sequencial de instruções. Vamos agora descrever mais alguns exemplos de problemas que podem ser resolvidos com a estrutura sequencial e implementados em `Python`. Para todos os exemplos, descrevemos a solução seguindo as Fases da metodologia proposta nas aulas 3 e 4.

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

1.1 Convertendo Temperaturas

As duas escalas mais usuais de temperatura são as escalas Celsius e Fahrenheit. O ponto de ebulição da água é 100° na escala Celsius e 212° na escala Fahrenheit. O ponto de congelamento da água é 0° na escala Celsius e 32° na escala Fahrenheit. Assumindo um relacionamento linear ($tempC = a \cdot tempF + b$) entre as duas escalas, projete um programa que converta uma temperatura dada na escala Fahrenheit para a temperatura correspondente na escala Celsius.

Para esse problema temos que a própria definição fornece praticamente um bom entendimento do problema.

```
Este algoritmo lê uma medida de temperatura na escala Fahrenheit e
e converte a temperatura para a escala Celsius e imprime o resultado
```

Vamos agora determinar o domínio e a imagem do problema (especificações da entrada e saída. Identificar a entrada e saída neste problema é fácil:

Entrada	**Saída**
Temperatura na escala Fahrenheit	Temperatura na escala Celsius

Calcular a conversão da temperatura na escala Fahrenheit para a escala Celsius pode ser feita facilmente de forma manual, ou mais facilmente ainda usando uma calculadora, e não justificaria o desenvolvimento de um programa para sua solução. O objetivo deste exemplo é o de ilustrar a utilização de expressões aritméticas num programa de computador.

Usamos variáveis do tipo `float` para armazenar as informações da temperatura na escala Fahrenheit (`tempF`) e da temperatura na escala Celsius (`tempC`). Considerando a relação linear ($tempC = a \cdot tempF + b$) entre as duas escalas e que

```
0.0 tempC == 32.0 tempF # congelamento
# substituindo na expressão tempC = a*tempF + b temos
0 == a*32.0 + b
b == -32.0*a
# e
100.0 tempC == 212.0 tempF # ebulição da água
# substituindo na expressão tempC = a*tempF + b temos
100.0 == a*212.0 + b
100.0 == a*212.0 -a*32.0
100.0 == a*180.0
a == 100.0/180.0
# e
b == -32.0*100.0/180.0
# substituindo a e b na expressão tempC = a*tempF + b temos
tempC = 100.0/180.0*tempF -32.0*100.0/180.0
tempC = 100.0/180.0*(tempF - 32.0)
```

e nesse caso, usando as variáveis `float`, temos a função

```
tempC(tempF) = 100.0/180.0 * (tempF - 32.0)
```

e as pré condições (domínio da função) e pós-condições (imagem) são as seguintes:

```
# pré: tempF

# pós: tempC == 100.0/180.0 * (tempF - 32.0)
```

O primeiro passo em um algoritmo para solucionar este problema é o de obter os valores para o item da entrada (input) - temperatura na escala Fahrenheit. A temperatura na escala Celsius pode então ser obtida pela subtração de 32.0 do valor da entrada `tempF` e depois multiplicar o valor resultante pela constante 100.0/180.0. Finalmente, o valor da saída (output) - temperatura na escala Celsius - deve ser impressa usando um dado formato.

O algoritmo abaixo ilustra uma solução para o problema de ler de uma temperatura na escala Fahrenheit, computar e imprimir a temperatura equivalente em na escala Celsius.

```
# Algoritmo: Converte
# Este algoritmo lê uma medida de temperatura na escala Fahrenheit,
# converte para a escala Celsius e imprime o resultado.
# descrição das variáveis locais utilizadas
# float tempF, tempC

# pré: tempF

# Passo 1. Leia a temperatura da entrada na escala Fahrenheit
# Passo 2. Calcule a temperatura equivalente na escala Celsius
# Passo 3. Imprima os resultados

# pós: tempC == 100.0/180.0 * (tempF - 32.0
# fim Algoritmo Converte
```

Os passos do Algoritmo que computa o valor da temperatura na escala Celsius para uma dada temperatura na escala Fahrenheit (Fases 1 a 3), descrevem como a entrada do problema (domínio) pode ser transformada na saída (imagem). Essa descrição tem que ser agora traduzida em instruções da linguagem Python.

Como vimos anteriormente, em `Python`, as variáveis podem ser de vários tipos (`int`, `float`, etc.) e os nomes das variáveis devem respeitar regras específicas para serem utilizadas, além disso, as variáveis não podem usar palavras reservadas da linguagem (`while`, `if`, etc.).

Considerando que o Python é uma linguagem multi-paradigma e utiliza tipos dinâmicos, não é necessário declarar as variáveis antes de sua utilização. Como uma estratégia de programação e documentação, vamos descrever, usando comentários, as variáveis que serão utilizadas, bem como seus respectivos tipos. Além disso, uma variável só pode ser referenciada se ela tiver sido inicializada, caso contrário, isso acarretará um erro no programa.

Após a descrição das variáveis (e constantes), temos que codificar a entrada e a saída do programa. A linguagem `Python` utiliza várias funções para a leitura da entrada. Inicialmente utilizaremos a função `input()` para a leitura e a função `print()` para impressão. Para utilizar essas funções não necessitamos incluir no programa nenhuma biblioteca.

Resolvido o problema da codificação da entrada e saída, falta computar o valor da temperatura na escala Celsius. Isso pode ser feito da seguinte forma:

```
tempC = 100.0/180.0 * (tempF - 32.0)
```

Vamos agora apresentar o programa completo. As fases 1 a 3 formam a parte da documentação do programa. Para leitura de dados (fluxo de entrada), temos a função `input()` e para a impressão (fluxo de saída) temos a função `print()`.

```
1  # -*- coding: utf-8 -*-
2  # Programa: convertFC.py
3  # Programador:
4  # Data: 16/05/2017
5  # Este programa lê uma medida de temperatura na escala Fahrenheit
6  # e converte para a escala Celsius e imprime o resultado.
7  # descrição das variáveis locais utilizadas
8  # início do módulo principal
9  # descrição das variáveis locais utilizadas
10 # float tempF, tempC
11
12 # pré: tempF
13
14 # Passo 1. Leia a temperatura da entrada na escala Fahrenheit
15 tempF = float(input())
16 # Passo 2. Calcule a temperatura equivalente na escala Celsius
17 tempC = 100.0/180.0 * (tempF - 32.0)
18 # Passo 3. Imprima os resultados
19 print('{0:5.1f} F = {1:5.1f} C'.format(tempF, tempC))
20
21 # pós: tempC == 100.0/180.0 * (tempF - 32.0
22 # fim do módulo principal
```

A primeira linha do programa

```
# -*- coding: utf-8 -*-
```

indica que a codificação dos caracteres no programa será a UTF-8. As linhas

```
# Passo 1. Leia a temperatura da entrada na escala Fahrenheit
tempF = float(input())
```

ilustram a leitura de valores `float` pelo programa. Esse formato é utilizado pelo Python 3.8. A função `input()` lê um fluxo de caracteres do dispositivo padrão de entrada e `float()` formata essa entrada como um número de ponto flutuante. A conversão da escala Fahrenheit para escala Celcius pode ser feita com

```
# Passo 2. Calcule a temperatura equivalente na escala Celsius
tempC = 100.0/180.0 * (tempF - 32.0)
```

onde a instrução busca na memória o valor armazenado na variável `tempF`, efetua, na unidade lógica e aritmética, as operações aritméticas que estão após o símbolo de atribuição "`=`" na expressão do Passo 2. Primeiro resolve a expressão dos parêntesis e depois, começando da esquerda para a direita do símbolo de atribuição, efetua a divisão e a multiplicação e atribui o resultado no endereço de memória relativo a variável `tempC`. Finalmente a instrução das linhas

```
# Passo 3. Imprima os resultados
print('{0:5.1f} F = {1:5.1f} C'.format(tempF, tempC))
```

ilustra a "impressão" no dispositivo padrão de saída da string

```
'{0:5.1f} F = {1:5.1f} C'
```

onde na posição `{0:5.1f}` será impresso o valor da variável `tempF` com cinco casas, sendo três para a parte inteira, uma para o "." e uma para a casa decimal. Na posição `{1:5.1f}` será impresso o valor da variável `tempC` com cinco casas, sendo três para a parte inteira, uma para o "." e uma para a casa decimal.

O programa pode ser interpretado/executado com o comando

```
$ python convertFC.py
```

e após ter corrigido as possíveis advertências e erros, a execução do programa para a entrada

```
# formato da entrada
32.0
```

terá como saída esperada:

```
# formato da saída
  32.0 F =   0.0 C
```

e para a entrada

```
# formato da entrada
100.0
```

a saída esperada será:

```
# formato da saída
100.0 F =  37.8 C
```

1.2 Perímetro e Área de um Triângulo Retângulo

Considere o problema de computar e imprimir o perímetro e a área de um triângulo retângulo, dados as medidas dos dois catetos. Nas Aulas 03 e 04 e descrevemos soluções para computar o perímetro e a área de um quadrado e de um círculo. Usando essas ideias, o programa `perimetroareaTR.py` descreve uma solução na linguagem `Python` para o problema de computar o perímetro e a área de um triângulo retângulo. Discutiremos os detalhes relativos dessa implementação, com destaque na implementação do Passo 2 do algoritmo. Como nas implementações anteriores na linguagem `Python`, a descrição do problema, as especificações de entrada e saída (as pré e pós-condições), as variáveis utilizadas e os passos do algoritmo são incluídas no programa como comentários, precedidos `#` e servem para *documentar* o programa.

```
#-*- coding: utf-8 -*-
# Programa: perimetroareaTR.py
# Programador:
# Data: 23/08/2020
# Este programa lê o valor dos catetos e de um triângulo retângulo e
# calcula e imprime o perímetro e a área do triângulo.
# declaração das bibliotecas utilizadas
import math
# início do módulo principal
# descrição das variáveis utilizadas
# float catetoop, catetoad, perimetro, area

# pré: catetoop, catetoad

# Passo 1. Leia os catetos do triângulo retângulo
print('Entre com os valores dos catetos do triângulo retângulo: ')
catetoop = float(input('Leia o valor do cateto oposto: '))
catetoad = float(input('Leia o valor do cateto adjacente: '))
# Passo 2. Calcule o perímetro e a área do triângulo
# Passo 2.1. Calcule o perímetro do triângulo retângulo
# Passo 2.1.1. Calcule o valor da hipotenusa
hipotenusa = math.sqrt(catetoop**2 + catetoad**2)
perimetro = catetoop + catetoad + hipotenusa
# Passo 2.2. Calcule a área do triângulo retângulo
area = (catetoop*catetoad)/2.0
# Passo 3. Imprima o perímetro e área
print('Perímetro = {0:.2f}'.format(perimetro))
print('Área = {0:.2f}'.format(area))

# pós: perimetro == catetoop + catetoad + hipotenusa and
#      area == (catetoop*catetoad)/2.0
# fim do módulo principal
```

Agora discutiremos mais alguns detalhes da implementação `Python` do programa acima. Como vimos no exemplo anterior, a função `input()` pode ser utilizada para ler a entrada do programa. As instruções abaixo

```
# Passo 1. Leia os catetos do triângulo retângulo
...
catetoop = float(input('Leia o valor do cateto oposto: '))
catetoad = float(input('Leia o valor do cateto adjacente: '))
```

leem dois números de ponto flutuante e os armazenam nas variáveis `catetoop` e `catetoad`. A função `input()` lê um fluxo de caracteres da entrada padrão (no nosso caso o teclado). Quando usamos `float(input())`, é feito a conversão do fluxo de caracteres para um número de ponto flutuante. Após a conversão os valores de ponto flutuante são atribuídos para as variáveis `catetoop` e `catetoad`.

Para calcular o perímetro, necessitamos os valores das medidas dos três lados do triângulo. Como o triângulo é retângulo, temos que calcular o valor da hipotenusa. Como temos

que $h^2 = op^2 + ad^2$, onde h é a hipotenusa, op é o cateto oposto e ad é o cateto adjacente, e os valores dos catetos são dados, temos que $h = \sqrt{op^2 + ad^2}$. Usando a biblioteca `math`, podemos calcular o valor da hipotenusa como:

```
# Passo 2.1.1. Calcule o valor da hipotenusa
hipotenusa = math.sqrt(catetoop**2 + catetoad**2)
```

e com isso, os cálculos do perímetro e da área do triângulo retângulo ficam:

```
# Passo 2. Calcule o perímetro e a área do triângulo
# Passo 2.1. Calcule o perímetro do triângulo retângulo
# Passo 2.1.1. Calcule o valor da hipotenusa
hipotenusa = math.sqrt(catetoop**2 + catetoad**2)
perimetro = catetoop + catetoad + hipotenusa
# Passo 2.2. Calcule a área do triângulo retângulo
area = (catetoop*catetoad)/2.0
```

Nas instruções abaixo, a função `print()`

```
# Passo 3. Imprima o perímetro e área
print('Perímetro = {0:.2f}'.format(perimetro))
....
```

imprime a string `'Perímetro = {0:.2f}'` usando o formado especificado `{0:.2f}`. Como vimos anteriormente, o primeiro campo, 0 é associado a primeira variável descrita em `format(...)`, `perimetro` e o segundo campo, `.2f` indica o tipo (`f==float`) e o formato como o número será exibido `.2f` indica que serão duas casas decimais após o "`.`". Para o valor de `lado == 5.3`, na impressão haverá a "substituição" de `{0:.2f}` por 5.30. A "impressão" final dessa instrução fica:

```
Perímetro = 5.30
```

Usando um Editor ou o IDLE (ambiente do Python), edite o programa e salve num diretório de trabalho com o nome `perimetroareaTR.py`. Verifique se você está no diretório onde o programa fonte `perimetroareaTR.py` foi salvo e execute o programa. Para interpretar/executar use o comando:

```
$ python perimetroareaTR.py
```

Os caminhos (`PATH`) para os módulos do Python devem estar devidamente definidos. O arquivo `perimetroareaTR.py` indica o programa fonte que será usado na interpretação/execução. Se você editou o programa corretamente a execução do comando de interpretação não gerará nenhuma advertência ou erro. Caso isso ocorra, veja o número da linha e verifique o que você digitou de forma errada. Após ter corrigido todas as possíveis advertências e erros, execute o comando novamente. Se não houver mais nenhum erro, a execução do programa começará.

Teste o seu programa para diversos valores dos lados de um triângulo retângulo, tais com `1.0` e `1.0`, e `3.0` e `4.0`. Os exemplos abaixo ilustram o resultado da execução do programa para essas entradas.

Exemplo 1:

```
# formato da entrada
Entre com os valores dos catetos do triângulo retângulo:
Leia o valor do cateto oposto: 1
Leia o valor do cateto adjacente: 1

# formato da saída
Perímetro = 3.41
Área = 0.50
```

Exemplo 2:

```
# formato da entrada
Entre com os valores dos catetos do triângulo retângulo:
Leia o valor do cateto oposto: 3
Leia o valor do cateto adjacente: 4

# formato da saída
Perímetro = 12.00
Área = 6.00
```

1.3 Total de Horas

Dado uma medida de tempo em semanas, dias e horas, como seriam os passos necessários para computar o total em horas dessa medida de tempo. A entrada é a quantidade de semanas, dias e horas e a saída é o total em horas dado pela quantidade de semanas multiplicada por 7×24 somado com a quantidade de dias multiplicado por 24 e somado com a quantidade de horas. Na Aula 03 descrevemos uma solução semelhante desse problema.

Abaixo descrevemos os detalhes do programa em `Python` que implementa essa solução. O importante é que os três principais passos são os mesmos que apareceram nas discussões das aulas. Cada um desses passos está claramente anotado como um comentário no programa (`#`. Os comentários em programação são úteis como instrumentos para descrever diretamente as ideias junto com o código do programa. Neste nosso curso vamos usá-los com este propósito.

```
1  # -*- coding: utf-8 -*-
2  # Programa: horas.py
3  # Programador:
4  # Data: 17/05/2016
5  # Este programa lê uma medida de tempo, dada em semanas,
6  # dias e horas e calcula seu equivalente em horas.
7  # início do módulo principal
8  # descrição das variáveis utilizadas
9  # int semanas -> representa o valor da entrada em semanas
10 # int dias -> representa o valor da entrada em dias
11 # int horas -> representa o valor da entrada em horas
12 # int totalhoras -> representa o valor total em horas
13
14 # pré: semanas, dias, horas
```

```
15
16 # Passo 1. Leia a entrada
17 print('Entre com a quantidade de semanas, dias, horas')
18 semanas = int(input())
19 dias = int(input())
20 horas = int(input())
21 # Passo 2. Calcule o total de horas
22 totalhoras = 7*24*semanas + 24*dias + horas
23 # Passo 3. Imprima o total de horas
24 print('{0:d} Semanas + {1:d} Dias + {2:d} Horas = {3:d} Horas'. \
25                             format(semanas, dias, horas, totalhoras))
26
27 # pós: totalhoras == 7*24*semanas+24*dias+horas
28 # fim do módulo principal
```

O caractere \ utilizado na função `print()` do `Passo 3` indica que a função continua na próxima linha.

Vamos agora descrever uma forma de como ler várias variáveis do mesmo tipo no `Python`. Como vimos nos exemplos anteriores, a função `input()` pode ser utilizada para ler a entrada do programa. Podemos combiná-la com outras instruções. A função `input()` nas instruções abaixo

```
# Passo 1. Leia a entrada
semanas, dias, horas = map(int, input().split())
```

lê três números inteiros separados por espaços e armazena-os nas variáveis `semanas`, `dias` e `horas`. A função `input()` lê uma string (fluxo de caracteres do dispositivo de entrada) da entrada padrão (no nosso caso o teclado), o método `split()` retorna a lista de todas as palavras da string. A função `map`, com o argumento `int` "converte" cada palavra da lista em um inteiro. Como o `Python` possibilita atribuição múltipla, os três inteiros são atribuídos às variáveis `semanas`, `dias` e `horas`, respectivamente. Caso a entrada seja a linha:

```
# formato da entrada
3 6 14
```

teremos `semanas == 3`, `dias == 6` e `horas == 14`.

Usando um Editor ou o ambiente IDLE do Python, edite o programa e salve num diretório de trabalho com o nome `horas.py`. Verifique se você está no diretório onde o programa fonte `horas.py` foi salvo e interprete/execute o programa. Para interpretar/executar use o comando:

```
$ python horas.py
```

Os caminhos (`PATH`) para as classes devem estar devidamente definidos. O arquivo `horas.py` indica o programa fonte que será usado na interpretação/execução. Se você editou o programa corretamente a execução do comando de interpretação não gerará nenhuma advertência ou erro. Caso isso ocorra, veja o número da linha e verifique o que você digitou de forma errada. Após ter corrigido as advertências e erros, execute o programa. O interpretador inicia a execução do programa.

A **entrada** pode ser uma medida de tempo qualquer, dada em semanas, dias e horas. Se os passos 1, 2 e 3 (**processo**) forem executados cuidadosamente a saída para a entradas dadas serão:
Exemplo 1:

```
# formato da entrada
3 6 14

# formato da saída
3 Semanas + 6 Dias + 14 Horas = 662 Horas
```

Exemplo 2:

```
# formato da entrada
5 5 15

# formato da saída
5 Semanas + 5 Dias + 15 Horas = 975 Horas
```

2 Problemas Computacionais com Texto

Vamos agora abordar problemas cuja solução computacional envolve palavras (strings). Cada conjunto de dados tem associado um conjunto de operações específicas, e no caso, a manipulação de strings é um pouco diferente da operação com conjuntos de dados numéricos. Veremos alguns detalhes referentes a manipulação de strings no Python.

2.1 Strings

Em aulas anteriores, usamos o tipo string sem uma definição adequada. O tipo *string* é importante em computação. Informalmente uma "string" é entendida como uma sequência qualquer de caracteres contíguos de um conjunto universal predefinido, conhecido como alfabeto. Como observamos anteriormente, um dos principais alfabetos usados pelos computadores é o conjunto de caracteres ASCII (American Standard Code for Information Interchange). Outros alfabetos existem, incluindo o EBCDIC para mainframes IBM. Aqui temos alguns exemplos de conjuntos cujos elementos são strings:

Exemplo 1 - O conjunto $\{0, 1, 00, 11, 01, 10, 111, \ldots\}$ é o conjunto de todas as possíveis strings de caracteres do alfabeto $\{0, 1\}$.

Exemplo 2 - $M = \{abc, acb, bac, bca, cab, cba\}$ é o conjunto de todas as strings que são um particular arranjo, ou permutação, das três letras a, b e c. Note que existem seis tais permutações.

Exemplo 3 - O conjunto dos genes de um dado organismo é um conjunto de strings sobre o alfabeto $\{A, T, C, G\}$.

Exemplo 4 - Um dicionário da língua portuguesa é um conjunto de strings sobre o alfabeto $\{a, b, c, \ldots, x, y, z\}$.

Nesta disciplina, quando tratando de strings, utilizaremos o conjunto de caracteres ASCII. Vários problemas trabalham com a manipulação de strings de caracteres ASCII, e suas soluções requerem o uso de um conjunto de operadores que possam manipular variáveis deste tipo. A tempos a computação reconheceu a importância das strings, mas não adotou nenhum padrão para a sua implementação. Em função disso, uma string criada numa dada linguagem pode ser diferente numa outra linguagem. Adotaremos a definição abaixo

Definição - Uma *string* é uma sequência finita de zero ou mais caracteres ASCII que sejam visíveis (para simplificar não consideraremos os caracteres estendidos). Quando utilizadas em um algoritmo devem estar entre aspas simples (`'`) para distingui-las das instruções do programa que estão junto. Quando armazenada, uma string ocupa n posições na memória. Dependendo da linguagem de programação é necessário uma posição a mais - n para os caracteres que compõe a string e um para o caractere nulo "caractere nulo" (`'\0 '`) que sinaliza o final da string. Por exemplo, a entrada

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

é uma string. Estendendo esta noção, podemos utilizar uma variável do tipo string ou um objeto do tipo string para armazená-la, dependendo da linguagem. As linguagens dispõem de várias funções ou métodos para manipular strings. Em alguns casos é conveniente copiar a string para um arranjo de caracteres ou transformá-la numa lista de subtrings. Nos exemplos anteriores, as variáveis que utilizamos para armazenar palavras são do tipo string.

Como estamos utilizando a linguagem `Python`, dispomos de um tipo string para armazenar um conjunto de zero ou mais caracteres. Além disso, o `Python` possui várias funções e métodos para manipular strings. Como o Python utiliza tipos dinâmicos, não é necessário declarar uma variável para armazenar uma string. De qualquer forma, sempre descrevemos as variáveis que utilizamos nos nossos programas.

Exemplos de variáveis do tipo string

```
# descrição das variáveis utilizadas
# str nome
# str sobrenome
# str texto
```

As operações que podemos fazer com o tipo de dados
tt string são diferentes das que podemos fazer com os tipos numéricos `int`, `float`, etc. Como vimos anteriormente, a atribuição de um texto a uma variável do tipo string na linguagem Python é feito com o operador de atribuição `=`.

De maneira análoga as várias operações que podemos fazer com valores numéricos, tais como soma, subtração, etc., com strings também podemos fazer várias operações.

2.2 Funções e Métodos para Strings no Python

Inicialmente vamos descrever algumas dessas operações. Normalmente essas operações para manipular strings já estão disponíveis e algum módulo da linguagem. Algumas dessas funções e métodos podem ser facilmente implementadas usando operações simples de manipulação dos caracteres da string.

Como vimos anteriormente uma string é dada por um conjunto de caracteres, cujo início e fim são sinalizados por `'` ou por `"`.

```
'Algoritmos'
```

A atribuição, leitura e impressão podem ser feitas com:

```
diciplina = input()
```

```
disciplina = 'Algoritmos'
```

```
print(disciplina)
print('Algoritmos')
```

O Python também possibilita que uma string seja dada em várias linhas:

```
textolatim = '''Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim
ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip
ex ea commodo consequat. Duis aute irure dolor in reprehenderit in
voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur
sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt
mollit anim id est laborum'''
```

onde o texto tem que estar sinalizado com `'''` ou por três aspas.

Como já vimos, uma string pode ser vista como um arranjo de caracteres, onde cada um dos caracteres pode ser acessado individualmente pelo índice do caractere na string.

```
disciplina = 'Algoritmos'
```

```
disciplina[0] = 'A'
disciplina[1] = 'l'
disciplina[2] = 'g'
disciplina[3] = 'o'
disciplina[4] = 'r'
disciplina[5] = 'i'
disciplina[6] = 't'
disciplina[7] = 'm'
disciplina[8] = 'o'
disciplina[9] = 's'
```

No Python, as strings são imutáveis, ou seja não é possível substituir um ou mais caracteres na string. Uma vez atribuída a uma variável, os valores dos caracteres não podem ser alterados.

Por outro lado, várias operações de *slicing* facilitam a operação com strings. O trecho de programa

```
disciplina = 'Algoritmos 1'
# imprimir os caracteres 3, 4, 5 e 6 da string disciplina
print(disciplina[3:7])
# saída do print()
orit
# indexação negativa
# imprimir os caracteres 3, 4, 5 e 6 da string disciplina
```

```
print(disciplina[-7:-3])
# saída do print()
orit
```

Vamos descrever algumas funções e métodos em `Python` que podem ser usadas na solução de problemas que envolvem strings. Algumas já foram usadas em exemplos anteriores.

```
disciplina = 'Algoritmos 1'
tam = len(disciplina)
print(len)
# saída do print()
12
```

o trecho de programa acima imprime o valor 12, que é o número de caracteres da string disciplina (inclusive o caracteres espaço).

Vamos agora analisar o método `strip()`. O método retira os caracteres espaço do início e do final da string:

```
disciplina = '  Algoritmos 1   '
print(disciplina.strip())
# saída do print()
Algoritmos 1
```

É possível usar esse método com parâmetros para retirar caracteres específicos do início e do final da string:

```
disciplina = '  Algoritmos e Estrutura 1   '
print(disciplina.strip(' glA 1ar'))
\textcolor{green}{# saída do print()}
\textcolor{magenta}{oritmos e Estrutu}
```

não é levado em consideração a ordem dos caracteres.

Os métodos `upper()` e `lower()` transformam os caracteres de uma string para maiúsculos e minúsculos, respectivamente.

```
disciplina = 'algoritmos'
print(disciplina.upper())
# saída do print()
ALGORITMOS
```

```
disciplina = 'ALGORITMOS'
print(disciplina.lower())
# saída do print()
algoritmos
```

Vamos agora descrever o método `replace()`. Este método substitui as ocorrências de uma dada string por outra string:

```
doce1 = 'bananada'
doce2 = doce1.replace('na','ta')
print(doce2)
# saída do print()
batatada
```

Na leitura de vários dados numa mesma linha, utilizamos o método `split()`. Esse método também pode ser utilizado para separar strings usando um dado separador. O método `str.split(separador,max)` retorna uma lista de `max` strings usando um dado `separador`.

```
disciplina = '  Algoritmos e Estrutura 1'
print(str.split())
# saída do print()
['Algoritmos', 'e', 'Estrutura', '1']
```

```
disciplina = 'Algoritmos e Estrutura 1'
print(str.split(' ',1))
# saída do print()
['Algoritmos', 'e Estrutura 1']
```

```
disciplina = 'maça, canela, pera, chocolate, fruta do conde'
print(str.split(,))
# saída do print()
['maça', 'canela', 'pera', 'chocolate', 'fruta do conde']
```

A análise da ocorrência de um padrão num dado texto pode ser feita de várias formas no `Python`. Uma delas é com a instruções `in` e `not in` para verificar se um *elemento* ocorre num dado conjunto. Nesse cado, dado uma string a verificação se um dado padrão ocorre na string pode ser feito com:

```
texto = 'A copa do mundo de futebol em 2014 foi no Brasil'
padrao = 'mundo' in texto
# saída do print()
print(padrao)
True
```

o trecho de programa acima imprime `True`, pois `mundo` ocorre no texto. Caso o padrão fosse `2018` a resposta seria `False`.

Caso necessitemos conhecer a posição onde um dado padrão ocorre no texto, podemos usar os métodos `find()` ou `index()`.

```
texto = 'A copa do mundo de futebol em 2014 foi no Brasil'
padrao = 'mundo'
pos = texto.find(texto)
# saída do print()
print(pos)
10
```

o resultado é a impressão do número 10, a posição onde o padrão ocorre no texto. Se usarmos o método `index()`, o resultado será o mesmo.

```
texto = 'A copa do mundo de futebol em 2014 foi no Brasil'
padrao = 'mundo'
pos = texto.index(texto)
# saída do print()
print(pos)
10
```

A diferença entre os métodos `find()` e `index()` é o retorno quando o padrão não for encontrado no `texto`. O método `find()` retorna -1 e o método retorna um exceção.

Outras duas operações que são efetuadas diretamente no `Python` são a concatenação de strings, que é feita com o operador `+` e a comparação de strings que é feita com os operadores lógicos usuais (`>`, `<`, `>=`, `<=` e `!=`.

Apesar de podermos usar esses métodos e comandos, todos esses problemas podem ser resolvidos manipulando individualmente os caracteres das strings. Na medida em que for sendo necessário, vamos apresentando novas funções e métodos para resolver problemas de manipulação de strings.

2.3 Lê e Imprime Palavra em Caracteres Maiúsculos

Nesse exemplo veremos como ler uma palavra usando o teclado do computador, transformá-la para caracteres maiúsculos e imprimir a palavra modificada na tela do computador. Para isso, temos que descrever alguns conceitos de manipulação de strings em `Python`. Inicialmente vamos considerar uma palavra como sendo um conjunto de caracteres contíguos sem o caractere espaço ou outro caractere de controle, como por exemplo o `enter`. O programa abaixo ilustra uma implementação em `Python`.

```
1  # -*- coding: utf-8 -*-
2  # Programa: leimprimeM.py
3  # Programador:
4  # Data: 22/10/2012
5  # Este algoritmo lê uma palavra, armazena numa variável, modifica
6  # seus caracteres para maiúsculos e imprime as duas palavras
7  # e imprime a palavra lida.
8  # início do módulo principal
9  # descrição das variáveis utilizadas
10 # string palavra, palavraM
11
```

```
12 # pré: palavra
13
14 # Passo 1. Leia uma palavra
15 palavra = input('Entre com uma palavra: ')
16 # Passo 2. Transforme a palavra para maiúsculos
17 palavraM = palavra.upper()
18 # Passo 3. Imprima a palavra com caracteres maiúsculos
19 print('{0:s} {1:s}'.format(palavra,palavraM))
20
21 # pós: palavra com todos os caracteres maiúsculos
22 # fim do módulo principal
```

O resultado do Passo 3 pode ser implementado em Python da seguinte maneira:

```
# Passo 3. Imprima as palavras
print(palavra,palavraM)
```

Usando um Editor ou o ambiente IDLE do Python, edite o programa e salve num diretório de trabalho com o nome `leimprime.py`. Verifique se você está no diretório onde o programa fonte `palavra.py` foi salvo e interprete/execute o programa. Para interpretar/executar use o comando:

```
$ python leimprime.py
```

Os caminhos (`PATH`) para as classes devem estar devidamente definidos. O arquivo `leimprime.py` indica o programa fonte que será usado na interpretação/execução. Se você editou o programa corretamente a execução do comando de interpretação não gerará nenhuma advertência ou erro. Caso isso ocorra, veja o número da linha e verifique o que você digitou de forma errada. Após ter corrigido as advertências e erros, execute o programa. O interpretador inicia a execução do programa.

Teste o seu programa para vários palavras e verifique se o programa está imprimindo corretamente as palavras lidas.

Exemplo 1:

```
# formato da entrada
Algoritmos

# formato da saída
Algoritmos ALGORITMOS
```

Exemplo 2:

```
# formato da entrada
Estrutura

# formato da saída
Estrutura ESTRUTURA
Estrutura
```

Exemplo 3:

```
# formato da entrada
Dados

# formato da saída
Dados DADOS
```

2.4 Copia Parte da Palavra 1

Projetar um programa que leia uma palavra com pelo menos 5 caracteres, o programa deve efetuar uma cópia dos três primeiros caracteres da palavra numa nova variável e imprime a palavra e a parte da palavra.

```
1  # -*- coding: utf-8 -*-
2  # Programa: copiappalavra.py
3  # Programador:
4  # Data: 22/10/2012
5  # Este algoritmo lê uma palavra com pelo menos 5 caracteres, faz uma
6  # cópia dos três primeiros caracteres da palavra numa nova variável
7  # e imprime a palavra e a nova variável
8  # início do módulo principal
9  # descrição das variáveis utilizadas
10 # string palavra, partepala
11
12 # pré: palavra
13
14 # Passo 1. Leia uma palavra
15 palavra = input('Entre com uma palavra: ')
16 # Passo 2. Faça uma cópia dos três primeiros caracteres da palavra lida
17 partepal = palavra[0:3]
18 # Passo 3. Imprima a palavra e a parte da palavra
19 print('palavra = {0:s} - parte = {1:s}'.format(palavra, partepal))
20
21 # pós: palavra partepal and partepal == palavra[0:3]
22 # fim do módulo principal
```

Como no exemplo anterior, o Passo 3 pode ser codificado como:

```
print('palavra = ' + palavra + ' - parte = ' + partepal)
```

Usando um Editor ou o ambiente IDLE do Python, edite o programa e salve num diretório de trabalho com o nome `copiappalavra.py`. Verifique se você está no diretório onde o programa fonte `copipalavra.py` foi salvo e interprete/execute o programa. Para interpretar/executar use o comando:

```
$ python copiapalavra.py
```

Os caminhos (PATH) para as classes devem estar devidamente definidos. O arquivo `copiappalavra.py` indica o programa fonte que será usado na interpretação/execução. Se você editou o programa corretamente a execução do comando de interpretação não gerará nenhuma advertência ou erro. Caso isso ocorra, veja o número da linha e verifique o que você digitou de forma errada. Após ter corrigido as advertências e erros, execute o programa. O interpretador inicia a execução do programa.

Teste o seu programa para vários palavras e verifique se o programa está imprimindo corretamente as palavras lidas. Os exemplos abaixo ilustram entradas e saídas para o problema,

Exemplo 1:

```
# formato da entrada
Algoritmos

# formato da saída
palavra = Algoritmos - parte = Alg
```

Exemplo 2:

```
# formato da entrada
Estrutura

# formato da saída
palavra = Estrutura - parte = Est
```

Exemplo 3:

```
# formato da entrada
Dados

# formato da saída
palavra = Dados - parte = Dad
```

2.5 Contando as ocorrências de um dado caractere numa Palavra

Projetar e implementar um programa que leia uma palavra e um caractere, o programa deve computar e imprimir o número de vezes que o caractere lido aparece na palavra. Para uma implementação em Python temos:

```
1 # -*- coding: utf-8 -*-
2 # Programa: numcaracteres.py
3 # Programador:
4 # Data: 16/08/2020
5 # Este programa lê uma palavra e um caractere, calcula e imprime o
6 # número de vezes que o caractere lido aparece na palavra.
7 # início do módulo principal
8 # descrição das variáveis utilizadas
```

```
#string palavra, caractere
#int numcaractere

# pré: palavra caractere

# Passo 1. Leia a entrada
# Passo 1.1. Leia uma palavra
palavra = input('Entre com uma palavra: ')
# Passo 1.2. Leia um caractere
caractere = input('Entre com um caractere: ')
# Passo 2. Calcule o número de vezes que caractere aparece na palavra
numcaractere = palavra.count(caractere)
# Passo 3. Imprima o numero de caracteres que aparecem na palavra
print('{0:d}'.format(numcaractere))

# pós: numcaractere == número de vezes que caractere aparece
#      na palavra)
# fim do módulo principal
```

No *Python* função `string.count(substring)` computa o número de vezes que `substring` aparece em `string`. Nesse exemplo, usamos a função `count()` para contar caracteres, pois no Python, um caractere também é considerado uma string.

Usando um Editor ou o ambiente IDLE do Python, edite o programa e salve num diretório de trabalho com o nome `numcaracteres.py`. Verifique se você está no diretório onde o programa fonte `numcaracteres.py` foi salvo e interprete/execute o programa. Para interpretar/executar use o comando:

```
$ python numcaracteres.py
```

Os caminhos (`PATH`) para as classes devem estar devidamente definidos. O arquivo `numcaracteres.py` indica o programa fonte que será usado na interpretação/execução. Se você editou o programa corretamente a execução do comando de interpretação não gerará nenhuma advertência ou erro. Caso isso ocorra, veja o número da linha e verifique o que você digitou de forma errada. Após ter corrigido as advertências e erros, execute o programa. O interpretador inicia a execução do programa.

Teste o seu programa para vários valores e verifique se o programa está computando corretamente o número de vezes que `caractere` aparace na palavra lida.

Exemplo 1:

```
# formato da entrada
Algoritmos
o

# formato da saída
2
```

Exemplo 2:

```
# formato da entrada
ola tudo bem
i

# formato da saída
0
```

Exemplo 3:

```
# formato da entrada
hahahahahahaha
h

# formato da saída
7
```

3 Problemas Computacionais com Ambiente Gráfico

Como vimos anteriormente, para solucionar problemas que envolvem imagens e gráficos, a linguagem *Python* possui várias bibliotecas. Nessa aula, usando a biblioteca `PIL`)`pillow`) mostraremos como gerar retas, retângulos e círculos. Vamos utilizar os recursos gráficos de módulos da biblioteca `PIL` do Python. Utilize o `IDLE` (ou uma IDE/Editor) para editar e executar o programas.

Nas aula anteriores, usando as bibliotecas `PIL e stddraw`, apresentamos como desenhar pontos e linhas numa janela gráfica. Na aula de hoje vamos descrever como combinar essas primitivas e como obter figuras geométricas.

3.1 Triângulo

Usando a biblioteca `math.h` vamos descrever como gerar e imprimir um triângulo retângulo de lado l numa janela gráfica de dimensão 900×600 pixels. Considerando que todos os lados l do triângulo são iguais, temos que calcular o valor da altura h do triângulo. Esse cálculo pode ser feito como:

$$h^2 + \frac{l}{2}^2 = l^2$$

$$h^2 + \frac{l^2}{4} = l^2$$

$$h^2 = l^2 - \frac{l^2}{4}$$

$$h^2 = \frac{3l^2}{4}$$

$$h = \sqrt{\frac{3l^2}{4}}$$

$$h = \frac{\sqrt{3}}{2}l$$

Usando o cálculo da altura, vamos desenhar um triângulo equilátero, onde cada lado tem comprimento de 300 pixels, e o vértice esquerdo inciando na posição $(300, 500)$.

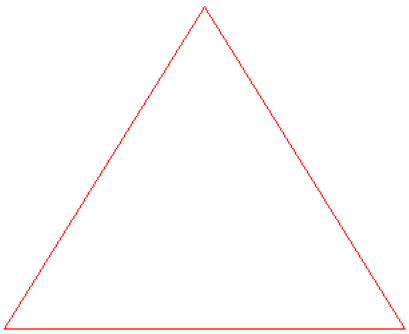

Figura 1: Triângulo Equilátero com 300 pixels de lado.

```
# -*- coding: utf-8 -*-
# Programa: triangulopil.py
# Programador:
# Data: 27/08/2020
# Este programa utiliza os módulos math e pillow.
# Ele gera e imprime um triângulo.
# Declaração dos módulos utilizados
from PIL import Image, ImageDraw
import math
# início do módulo principal
# descrição das variáveis utilizadas
# float altura
# Image tela
# ImageDraw desenha

# pré: void

# Passo 1. Defina o ambiente gráfico
# Passo 1.1. Defina o tamanho da imagem/tela
largura = 900
altura = 600 # 200   # 800
# Passo 1.2. Defina o padrão de cores
padrao = 'RGB'
```

```
# Passo 1.3. Crie a imagem/tela
tela = Image.new(padrao, (largura,altura), color='white')
desenhe = ImageDraw.Draw(tela)
# Passo 2. Gere um triângulo equilátero de 300 pixels de lado
# Passo 2.1. Desenhe uma linha (base do triângulo)
desenhe.line((300, 500, 600, 500),fill=(255, 0, 0), width=1)
# Passo 2.2. Compute a altura do triângulo
altura = (math.sqrt(3.0)/2.0)*300
# Passo 2.3. Desenhe uma linha (lado)
desenhe.line((300, 500, 450, altura),fill=(255, 0, 0), width=1)
# Passo 2.4. Desenhe uma linha (lado)
desenhe.line((450, altura, 600, 500),fill=(255, 0, 0), width=1)
# Passo 3. Imprima o triângulo
tela.show()

# pós: um triângulo
# fim do módulo principal
```

A compilação/interpretação deve ser feita com:

```
$ python triangulo.py
```

O programa depende dos módulos `stddraw.py` e `math.h`.

(a) Modifique o programa para desenhar um quadrado.

(b) Modifique o programa para desenhar um trapézio.

3.2 Figuras

Como vimos nas Aulas 03 e 04, o Python possui várias bibliotecas com funções que auxiliam e simplificam a manipulação dos recursos gráficos do Python. Neste programa vamos descrever a utilização de módulos da biblioteca `pillow` (`PIL`). Vamos usar os módulos `Image` e `ImageDraw`. O módulo `Image` possibilita criar uma imagem (tela) onde podemos incorporar (desenhar) vários componentes. O módulo `ImageDraw` tem vários componentes que podem ser adicionadas a imagem/tela criada. Neste exemplo vamos usar o `line()`, o `ellipse()` e o `rectangle()`. Para ilustrar essas funções/métodos, abaixo descrevemos um programa que imprime reta ligando um círculo e um retângulo usando os módulos acima.

```
# -*- coding: utf-8 -*-
# Programa: circlerect.py
# Programador:
# Data: 23/08/2020
# Este programa utiliza o módulo pillow (PIL).
# Ele gera um círculo, um retângulo e uma linha lingando as figuras
# e imprime as figuras.
# Declaração dos módulos utilizados
from PIL import Image, ImageDraw
# início do módulo principal

```

```
12 # pré: void
13
14 # Passo 1. Defina o ambiente gráfico
15 # Passo 1.1. Defina o tamanho da imagem/tela
16 largura = 500
17 altura = 300
18 # Passo 1.2. Defina o padrão de cores
19 padrao = 'RGB'
20 # Passo 1.3. Crie a imagem/tela
21 tela = Image.new(padrao, (largura,altura), color='white')
22 desenhe = ImageDraw.Draw(tela)
23 # Passo 2. Gere as figuras
24 # Passo 2.1. Gere um círculo de centro (75,50) e raio 25
25 desenhe.ellipse([(50,50),(100,100)],fill=(0,255,0))
26 # Passo 2.2. Desenhe um retângulo em (150,50) larg 100 e alt 50
27 desenhe.rectangle([(150,50),(250,100)],fill=(255,255,0))
28 # Passo 2.3. Desenhe uma linha ligando as duas figuras
29 desenhe.line([(100,75),(150,75)],fill=(255,0,0))
30 # Passo 3. Imprima os componentes
31 tela.show()
32
33 # pós: um círculo e um retângulo conectados por uma reta
34 # fim do módulo principal
```

A interpretação/compilação pode ser feita com:

```
$ python circlerect.py
```

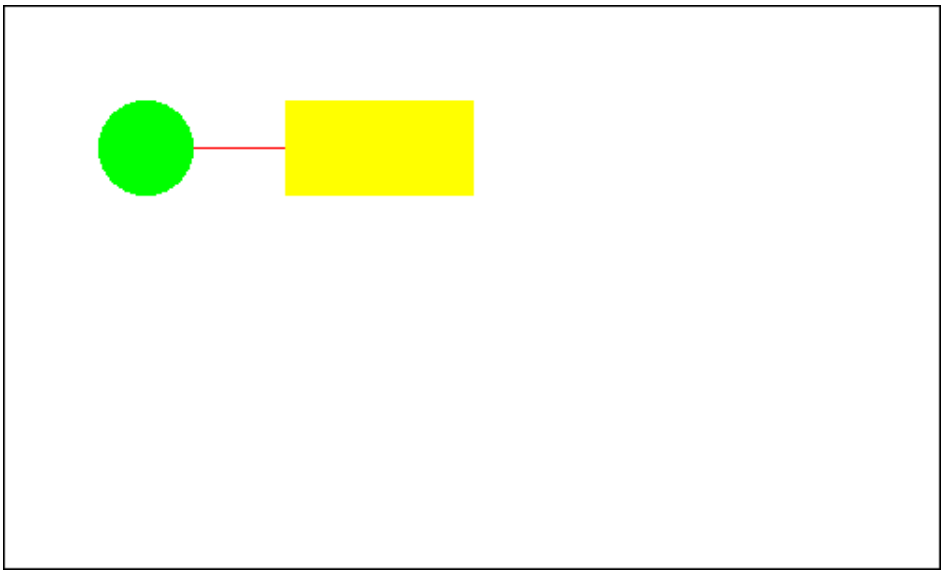

Figura 2: Saída do Programa `circlerect.py`.

A instrução

```
# Passo 2.1. Gere um círculo de centro (75,50) e raio 25
desenhe.ellipse([(50,50),(100,100)],fill=(0,255,0))
```

gera um círculo de centro $(75, 50)$ e raio 25 numa janela gráfica. O círculo "inserido" no quadrado $[(50, 50), (100, 100)]$ (região delimitante). A figura abaixo ilustra a posição das coordenadas,

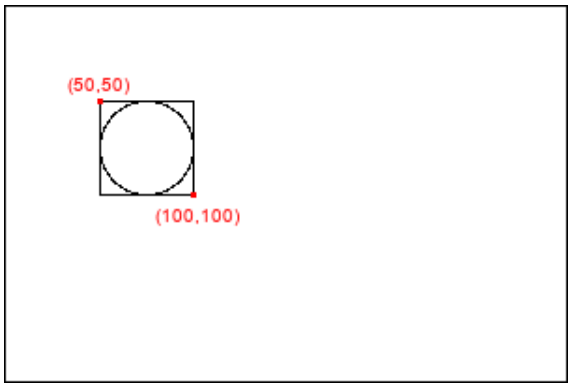

Figura 3: Coordenadas do Círculo.

e a instrução

```
# Passo 2.2. Desenhe um retângulo em (150,50) larg 100 e alt 50
desenhe.rectangle([(150,50),(250,100)],fill=(255,255,0))
```

gera um retângulo com o vértice superior esquerdo em $(150, 50)$ e vértice inferior direito em $(250, 100)$. A seguir, a instrução

```
# Passo 2.3. Desenhe uma linha ligando as duas figuras
desenhe.line([(100,75),(150,75)],fill=(255,0,0))
```

gera uma reta entre as bordas do círculo $((100, 75))$ e a do retângulo $((150, 75))$ e finalmente, a instrução

```
# Passo 3. Imprima os componentes criados
tela.show()
```

mostra os objetos criados numa janela gráfica (tela).

A execução desse programa necessita que a biblioteca `pillow` (`PIL`) esteja instalada. Utilize a sua criatividade para gerar outras figuras, mudar as cores e explorar os módulos `Image` e `ImageDraw`. Na próximas aulas abordaremos outras funcionalidades da biblioteca `PIL`.

4 Exercícios

4.1 Conversão de Temperatura - Celsius para Fahrenheit

As duas escalas mais usuais de temperatura são as escalas Celsius e Fahrenheit. O ponto de ebulição da água é 100° na escala Celsius e 212° na escala Fahrenheit. O ponto de congelamento da água é 0° na escala Celsius e 32° na escala Fahrenheit. Assumindo um relacionamento linear ($tempC = a \cdot tempF + b$) entre as duas escalas, projete um programa que converta uma temperatura dada na escala Celsius para a temperatura correspondente na escala Fahrenheit. Use o Exemplo 1 para obter a equação para transformar temperatura na escala Celsius para escala Fahrenheit. O formato da saída do programa é dada por:

```
# Passo 3. Imprima os resultados
print('{0:5.1f} C = {1:5.1f} F'.format(tempC, tempF))
```

Exemplo 1:

```
# formato da entrada
0.0

# formato da saída
  0.0 C =  32.0 F
```

Exemplo 2:

```
# formato da entrada
37.8

# formato da saída
 37.8 C = 100.0 F
```

4.2 Perímetro e Área de um Triângulo Isósceles

Usando o Exemplo 2 da Seção 1, e a subdivisão abaixo, projete e implemente um programa em `Python` para ler os lados de um triângulo isósceles, calcular e imprimir o perímetro e a área do triângulo. A entrada é dada por três valores do tipo `float`, representando a base (`base`) e os dois lados iguais (`lado1`) e `lado2`). A saída consiste em imprimir o perímetro e a área com um formato específico.

```
1  # Programa: perimetroareaTI.py
2  # Este programa lê o valor dos lados de um triângulo isósceles e
3  # calcula e imprime o perímetro e a área do triângulo.
4  # início do módulo principal
5  # descrição das variáveis utilizadas
6  double base, lado1, lado2, perímetro, área
7
8  # pré: base, lado1, lado2
9
10 # Passo 1. Leia a base e os dois lados iguais do triângulo
11 # Passo 2. Calcule o perímetro e a área do triângulo isósceles
12 # Passo 2.1. Calcule o perímetro do triângulo
13 # Passo 2.2. Calcule a área do triângulo
14 # Passo 3. Imprima o perímetro e a área do triângulo
15 print('Perímetro = {0:.2f}'.format(perimetro))
16 print('Área = {0:.2f}'.format(area))
17
18 # pós: perimetro == base+lado1+lado2 and area == (base*altura)/2
19 # fim do módulo principal
```

Exemplos de entrada e saída que o programa deve atender:

Exemplo 1:

```
# formato da entrada
6 # base
5 # medida dos lados iguais

# formato da saída
Perímetro = 16.00
Área = 12.00
```

Exemplo 2:

```
# formato da entrada
3 # base
2 # medida dos lados iguais

# formato da saída
Perímetro = 7.00
Área = 1.98
```

4.3 Copia Parte da Palavra 2

Usando o Problema 1 da Seção 2, e a subdivisão abaixo, projete e implemente um programa em `Python` que leia uma palavra com pelo menos 5 caracteres, faz uma cópia dos três últimos caracteres da palavra numa nova variável e imprime a palavra e a parte da palavra.

```
1  # -*- coding: utf-8 -*-
2  # Programa: copiappalavra2.py
3  # Programador:
4  # Data: 22/10/2012
5  # Este algoritmo lê uma palavra com pelo menos 5 caracteres, faz uma
6  # dos três últimos caracteres da palavra numa nova variável
7  # e imprime a palavra e a nova variável
8  # início do módulo principal
9  # descrição das variáveis utilizadas
10 # string palavra, partepala
11
12 # pré: palavra
13
14 # Passo 1. Leia uma palavra
15 # Passo 2. Faça uma cópia dos três últimos caracteres da palavra lida
16 # Passo 3. Imprima a palavra e a parte da palavra
17 print('palavra = {0:s} - parte = {1:s}'.format(palavra, partepal))
18
19 # pós: palavra partepal and partepal == palavra[tam-3:tam]
20 # fim do módulo principal
```

Teste o seu programa para vários palavras e verifique se o programa está imprimindo corretamente as palavras lidas. Os exemplos abaixo ilustram entradas e saídas para o problema,

Exemplo 1:

```
# formato da entrada
Algoritmos

# formato da saída
palavra = Algoritmos - parte = mos
```

Exemplo 2:

```
# formato da entrada
Estrutura

# formato da saída
palavra = Estrutura - parte = ura
```

Exemplo 3:

```
# formato da entrada
Dados

# formato da saída
palavra = Dados - parte = dos
```

4.4 Procurando Substrings numa Sequência de DNA

O exercício a seguir explora a busca por substrings numa sequência de DNA. Em Biologia Molecular Computacional, uma sequência de DNA é uma sequência formada com caracteres no alfabeto {A,T,C,G}, representando os quatro nucleotídeos – as bases adenina, citosina, guanina, timina (Sequência de Ácido Nucleico). A análise de trechos dessa sequência (substring) possuem propriedades biológicas. Uma delas é número de *tandem repeats* que é bastante usado em Biologia Computacional e Computação Forense, pois podem ser usados para verificar se uma dada amostra tem relação com o DNA de uma pessoa. Nossos exemplos não possuem significado biológico e as sequências e subsequências aqui descritas foram geradas de forma aleatória. Dadas as substrings:
CTA, GATA e AGAC podemos observar como elas ocorrem na sequência abaixo:

```
ATCGCTACTACTACATGCTACTAGAGACAGACAGACAAAGGTTCGATAGATAGATAGATATTAA
```

No caso do `tandem repeats`, o que se procura é o maior número de repetições de cada uma das substrings. Nesse exemplo, temos que CTA repete 3 vezes e depois 2 vezes. Consideramos o maior valor, 3. No caso das substrings GATA e AGAC elas repetem 4 e 3, respectivamente.

Ainda não temos as ferramentas computacionais adequadas para computar os maiores *tandem repeats*. Um passo inicial da solução é saber onde as substrings ocorrem e verificar se elas se repetem e o número de repetições. Um outro problema interessante é saber o número de vezes que as substrings ocorrem numa dada string. Neste exercício desejamos computar o

número de ocorrências de três substrings numa dada string de entrada, independentemente do fato de que as substrings repitam ou não. A entrada é dada por uma string e três substrings e desejamos computar e imprimir o número de vezes que cada uma das substrings ocorre na string.

Usando o Problema 3 da Seção 2, e a subdivisão abaixo, projete e implemente um programa para ler uma string e três substrings, calcular o número de vezes que cada uma das substrings aparece na string, e imprimir o resultado. Como vimos no Problema 3, podemos usar a função `count()` do `Python` para computar o número de vezes que uma substring aparace numa sequência de DNA.

```
1  # -*- coding: utf-8 -*-
2  # Programa: numsubstrings.py
3  # Este programa lê uma string e três substrings, calcula e imprime
4  # o número de vezes que cada substring aparece na string.
5  # início do módulo principal
6  # descrição das variáveis utilizadas
7  # string sequencia, subseq1, subseq2, subseq3
8  # int numsubseq1, numsubseq2, numsubseq3
9
10 # pré: sequencia subseq1 subseq2 subseq3
11
12 # Passo 1. Leia uma sequência e 3 subsequências
13 # Passo 2. Compute o num de vezes que cada subseq aparece
14 # Passo 3. Imprima o num de vezes que cada subseq aparace
15
16 # pós: numsubseq1 = sequencia.count(subseq1) and
17 #      numsubseq2 = sequencia.count(subseq2) and
18 #      numsubseq3 = sequencia.count(subseq3)
19 # fim do módulo principal
```

Abaixo descrevemos uma implementação do Passo 3 para auxiliar na solução.

```
# Passo 3. Imprima o num de vezes que cada subseq aparaces
print('{0:s} - {1:d}'.format(subseq1,numsubseq1))
print('{0:s} - {1:d}'.format(subseq2,numsubseq2))
print('{0:s} - {1:d}'.format(subseq3,numsubseq3))
```

Teste o seu programa para diversos conjuntos de duas palavras:

Exemplo 1:

```
#  formato da entrada
ACTGCTACTACTACATGCTACTAGAGACAGACAGACAAAGGTTCGATAGATAGATAGATATTAA
CTA
GATA
AGAC

# formato da saída
CTA - 5
GATA - 4
AGAC - 3
```

Exemplo 2:

```
#  formato da entrada
ACTGATACTAAGACCTAAGATGCTAGAGACAGACAGACAAAGGTTCGATACTATGTTAGACATAA
CTA
GATA
AGAC

# formato da saída
CTA - 4
GATA - 2
AGAC - 5
```

4.5 Pizza

Usando o Iddle ou um editor de programas, projete e implemente um programa para desenhar uma pizza com 4 pedaços na tela do computador. O programa deve usar a biblioteca `PIL`, e usar uma janela gráfica (tela) com dimensões 900×600, e a pizza deve estar centrada no ponto $(600, 300)$ da janela gráfica e ter raio 200 (pixels). Use o esboço abaixo para finalizar o seu programa.

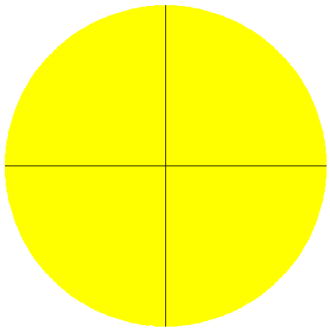

Figura 4: Saída do Programa `pizza.py`.

```
# -*- coding: utf-8 -*-
# Programa: pizza.py
# Programador:
# Data: 26/08/2020
# Este programa utiliza a biblioteca PIL (pillow).
# Ele gera um círculo e duas retas perpendiculares que passam pelo
# seu centro e terminam nas suas bordas, e imprime as figuras.
# Declaração dos módulos utilizados
from PIL import Image, ImageDraw
# início do módulo principal
# descrição das variáveis utilizadas
# int largura, altura
```

```
# str padrao

# pré: void

# Passo 1. Defina o ambiente gráfico
# Passo 1.1. Defina o tamanho da imagem/tela
largura = 900
altura = 600
# Passo 1.2. Defina o padrão de cores
padrao = 'RGB'
# Passo 1.3. Crie a imagem/tela
tela = Image.new(padrao, (largura,altura), color='white')
desenhe = ImageDraw.Draw(tela)
# Passo 2. Gere a pizza
# Passo 2.1. Gere um círculo de centro (600,300) e raio 200
# Passo 2.2. Desenhe uma linha entre (600,100) e (600,500)
# Passo 2.3. Desenhe uma linha entre (400,300) e (800,300)
# Passo 3. Imprima a pizza

# pós: uma pizza
# fim do módulo principal
```

Depois que você gerou a pizza, tente "colocar" uma "fatia de tomate" em cada pedaço da pizza. Tenta também "colocar" algumas "folhas de manjericão" na pizza.