Módulo 2: *Loops*, Funções

Objetivos de aprendizagem:

1. Entender a estrutura dos *loops* `for` e `while` e a diferença entre iteração em relação ao `index` versus `elemento`. Reconhecer os efeitos do `break` e `continue` `keywords`.
2. Construir novas funções e fazer chamadas de funções. Compreender o papel dos parâmetros.

2.1: *Loops*

2.1.1: *Loops For/While*

O Python tem uma implementação/estrutura do *loop* `FOR` que funciona tanto para o *loop* `for` padrão quanto para o *loop* `for-Each` de outras linguagens de programação.

Existem algumas diferenças estruturais no loop `FOR` do Python. Considere-as nos exemplos abaixo.

In [ ]:
```
"""
A variável **i** serve como o contador sobre RANGE de [0,10),
inclusive do limite inferior, mas exclusive do limite superior.

O limite inferior de 0 NÃO precisa ser especificado;
ele é implementado por padrão, a menos que outro limite inferior seja especificado.

Também, por padrão, se NÃO existir um terceiro parâmetro para range(),
então **i** aumenta em 1.
"""
for i in range(10):
  print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

In [ ]:
```
"""
Estes exemplos especificam um limite inferior que difere do valor padrão de 0.

A ordem do parâmetro é SEMPRE:
 1. Posição inicial, inclusive
 2. Posição de parada, exclusive
 3. Valor incremental


Neste exemplo, x começa com o valor igual a 2 e para em 9 inclusive, ou seja, 10 excl
usive,
com x acréscimos de 1 para cada iteração.
"""
for x in range(2, 10):
  print(i)
```

```
9
9
9
9
9
9
9
9
```

In [ ]:
```
"""
O terceiro parâmetro range() define o número de etapas para incrementar o contador.

Neste exemplo, x começa com o valor igual a 0 e para em 9 inclusive, ou 10 exclusive,
com x incrementos de 3 para cada iteração.
"""

for i in range(0, 10, 3):
  print(i)
```

```
0
3
6
9
```

In [ ]:
```
"""
Loop 'for-Each' sobre cada caractere em uma string.

Neste exemplo, a variável 'i' representa cada elemento/caractere de 'hello'.
"""

for i in "hello!":
  print(i)
```

```
h
e
l
l
o
!
```

In [ ]:
```
"""
Poderíamos também iterar sobre a string por indexação.

Considere o seguinte exemplo de iteração sobre a string por index,
começando no índex 0 e terminando no último elemento, com os incrementos do contador
 em 2,
portanto, imprimindo APENAS todos os outros elementos da string.
"""

string = "hello world!"
for i in range(0, len(string), 2):
  print(str(i) + "th letter is "+ string[i])
```

```
0th letter is h
2th letter is l
4th letter is o
6th letter is w
8th letter is r
10th letter is d
```

A estrutura do *loop* `WHILE` permanece em sua maioria idêntica a outras linguagens de programação.

Dê uma olhada no exemplo abaixo

In [ ]:
```
#  Nota: sem a atualização da variável no loop while, este será um loop INFINITO!!

count = 0
while (count < 10):
  print(count)

  # IMPORTANTE!!! Atualize o contador!
  count += 1
```

```
0
1
2
3
4
5
6
7
8
9
```

2.1.2: *Continue/Break Keywords*

Como manipular *loops* (`FOR` e `WHILE`):

- **Break**: pular os códigos restantes no loop bem como as iterações remanecentes, quebrar (break) o loop mais interno.
- **Continue**: pular os códigos restantes no loop e continuar para a próxima iteração do loop.

In [ ]:
```
# Considere um programa que ecoe o input do usuário, exceto pelo "end".
# Este programa roda infinitamente, exceto quando o usuário insere "end" para terminá
-lo.

while True:
  user = input("Enter something to be repeated: ")

  ## Quando o "break" é acionado, o print() abaixo NÃO será executado.
  ## O programa romperá o loop quando esta palavra-chave for lida.
  if user=="end":
    print("Terminate the program!!!")
    break
  print(user)
```

```
Enter something to be repeated: a
a
Enter something to be repeated: g
g
Enter something to be repeated: end
Terminate the program!!!
```

In [ ]:
```
# Sem usar a palavra-chave "break", esta é outra implementação do mesmo programa de c
ima usando uma variável.

end = False
while end == False:
  user = input("Enter something to be repeated: ")
  if user=="end":
    print("Program Ended!!!")
    end = True
  else:
    print(user)
```

```
Enter something to be repeated: a
a
Enter something to be repeated: f
f
Enter something to be repeated: h
h
Enter something to be repeated: d
d
Enter something to be repeated: end
Program Ended!!!
```

In [ ]:

```
"""
Vamos considerar um loop que conta de 1-20, mas ignora todos os números que são múlti
plos de 5.
Neste caso, NÃO poderíamos usar a palavra-chave "break", porque isso encerrará o loo
p.
Queremos "continuar" o loop, exceto por alguns números.

Vamos implementar isto tanto com um loop "while" quanto com um loop "for".
"""

count = 1

# Implementação do loop WHILE
while count + 1 <= 20:
  if count % 5 == 0:
    print("SKIP")
    count += 1
    continue
  print(count)
  count += 1
```

```
1
2
3
4
SKIP
6
7
8
9
SKIP
11
12
13
14
SKIP
16
17
18
19
```

In [ ]:
```
# Implementação do loop FOR

for i in range (1, 20):
  if i % 5 == 0:
    print("SKIP")
    continue
  print(i)
```

```
1
2
3
4
SKIP
6
7
8
9
SKIP
11
12
13
14
SKIP
16
17
18
19
```

2.2: Funções

Funções são úteis quando lhe é dado um problema que pode ser dividido em várias etapas e algumas etapas são usadas de forma repetitiva. Então, ter uma função para essas etapas é conveniente porque reduz a repetição de código e torna o código mais organizado.

Notas sobre a definição de uma nova função:

1. Definir/inicializar uma nova função com a palavra-chave `def` antes do nome da função.
2. NÃO definir o tipo de retorno na declaração da função.
3. NÃO esquecer do parâmetro da função se sua função precisar de informações do `main()` ou de outras funções.
4. A declaração RETURN não é necessária, dependendo das funções.

In [ ]:

```
"""
Escreveremos nossa própria função que testa se um triângulo de 3 lados é um triângulo retângulo.
Como não podemos controlar a ordem dos lados que o usuário nos dá (de tal forma que C é o comprimento mais longo),
precisamos verificar manualmente se C é o comprimento mais longo (os comprimentos A e B podem estar em qualquer ordem).
Caso contrário, nosso teorema de Pitágoras falhará.
"""

def isRightTriangle(a, b, c):

  # Reatribuir valores variáveis para garantir que C seja o comprimento mais longo
  if (max(a,b,c) != c):

    # tmp armazena os valores anteriores de C, que não é o comprimento mais longo
    tmp = c
    c = max(a,b,c)

    if a == c:
      a = tmp
    elif b == c:
      b = tmp


  # Aplicar a fórmula
  if a**2 + b**2 == c**2:
    print("Right Triangle")

    # Se o programa vê uma declaração Return, este é o FIM do programa/função
    return True

  # Estas duas linhas funcionarão SOMENTE quando a condição IF for falsa
  print("NOT a right triangle")
  return False


# Solicite ao usuário que insira 3 comprimentos
def main():
  a = input("Enter the length for the first edge of the triangle:")
  b = input("Enter the length for the second edge of the triangle:")
  c = input("Enter the length for the last edge of the triangle:")

  # As entradas do usuário são armazenadas como uma string, então nós as computamos para ser int
  return isRightTriangle(int(a), int(b), int(c))

if __name__ == "__main__":
    main()
```

```
Enter the length for the first edge of the triangle:5
Enter the length for the second edge of the triangle:4
Enter the length for the last edge of the triangle:3
Right Triangle
```

In [ ]:

```
"""
Outro exemplo: determinar se a entrada do usuário é um palíndromo.

Palíndromo: se uma palavra/sentença é soletrada da mesma maneira quando é invertida.
  EX: racecar (carro de corrida)

Para este exemplo, vamos tornar isto de forma mais abrangente.
Em vez de verificar se uma palavra é um palíndromo, vamos testar se uma frase é um pa
líndromo.

A fim de escrever este programa, estabeleceremos algumas especificações:
  - Tratar as letras maiúsculas como minúsculas
  - Ignorar todos os espaços brancos e pontuações
  - Uma sentença/string vazia é considerada como um palíndromo.
"""

# importar o pacote de string
# Revisaremos mais pacotes/bibliotecas no Módulo 5
import string


# Esta implementação da função RETURN (retornar) um valor booleano, True/False (Verda
deiro/Falso)
def isPalindrome(str):

   # Como não conseguimos controlar o que o usuário insere na sentença, vamos esclare
cer primeiro a sentença.
   # Vamos remover todas as pontuações e espaços brancos da sentença e colocar todas
 as letras em minúsculo
   exclude = set(string.punctuation)
   str = ''.join(ch for ch in str if ch not in exclude)
   str = str.replace(" ", "").lower()

 # Comparar a string original com a string em ordem inversa
    # Observação de str[::-1]: os dois primeiros números definem o início e o fim da
 string, o último número de -1 indica a ordem inversa

  # Verificar se a string é a mesma tanto em ordem invertida quanto na ordem original
  if str == str[::-1]:
    return True
  else:
    return False

# Solicitar ao usuário que introduza a sentença
def main():
  userSentence = input("Enter a sentence to be tested as a palindrome:")

  if (isPalindrome(userSentence)):
    print(userSentence + " is a palindrome!")
  else:
    print(userSentence + " is NOT a palindrome!")

if __name__ == "__main__":
    main()
```

```
Enter a sentence to be tested as a palindrome:racecar
racecar is a palindrome!
```

In [ ]:
```
# Considere esta implementação da função que RETURN (retorna) uma string.
# Anote a diferença entre main() e isPalindrom() após esta mudança.

import string

def isPalindrome(str):
  exclude = set(string.punctuation)
  str = ''.join(ch for ch in str if ch not in exclude)
  str = str.replace(" ", "").lower()
  if str == str[::-1]:
    return str + " is a palindrome!"
  else:
    return str + " is NOT a palindrome!"

def main():
  userSentence = input("Enter a sentence to be tested as a palindrome:")
  print(isPalindrome(userSentence))

if __name__ == "__main__":
    main()
```

In [ ]:
```
"""
Acima trabalhamos com um exemplo que testa se uma sentença é um palíndromo.
Agora é a sua vez.

Exercício: escreva uma função para testar se uma palavra do usuário é um palíndromo.

As declarações de função são fornecidas para você.
"""

def isPalindrome(str):



# Solicitar que o usuário digite a sentença
def main():
  userInput = input("Enter a WORD to be tested as a palindrome:")

  if (isPalindrome(userInput)):
    print(userInput + " is a palindrome!")
  else:
    print(userInput + " is NOT a palindrome!")

if __name__ == "__main__":
    main()
```

NÃO VEJA A SOLUÇÃO ABAIXO ANTES DE TENTAR FAZER O EXERCÍCIO!

NÃO VEJA A SOLUÇÃO ABAIXO ANTES DE TENTAR FAZER O EXERCÍCIO!

NÃO VEJA A SOLUÇÃO ABAIXO ANTES DE TENTAR FAZER O EXERCÍCIO!

NÃO VEJA A SOLUÇÃO ABAIXO ANTES DE TENTAR FAZER O EXERCÍCIO!

NÃO VEJA A SOLUÇÃO ABAIXO ANTES DE TENTAR FAZER O EXERCÍCIO!

NÃO VEJA A SOLUÇÃO ABAIXO ANTES DE TENTAR FAZER O EXERCÍCIO!

NÃO VEJA A SOLUÇÃO ABAIXO ANTES DE TENTAR FAZER O EXERCÍCIO!

NÃO VEJA A SOLUÇÃO ABAIXO ANTES DE TENTAR FAZER O EXERCÍCIO!

NÃO VEJA A SOLUÇÃO ABAIXO ANTES DE TENTAR FAZER O EXERCÍCIO!

NÃO VEJA A SOLUÇÃO ABAIXO ANTES DE TENTAR FAZER O EXERCÍCIO!

NÃO VEJA A SOLUÇÃO ABAIXO ANTES DE TENTAR FAZER O EXERCÍCIO!

NÃO VEJA A SOLUÇÃO ABAIXO ANTES DE TENTAR FAZER O EXERCÍCIO!

NÃO VEJA A SOLUÇÃO ABAIXO ANTES DE TENTAR FAZER O EXERCÍCIO!

NÃO VEJA A SOLUÇÃO ABAIXO ANTES DE TENTAR FAZER O EXERCÍCIO!

In [ ]:

```
# Uma implementação de solução para o exercício acima.

def isPalindrome(str):
  str = str.lower()
  if (str == str[::-1]):
    return True
  else:
    return False


# Solicitar ao usuário que introduza a sentença
def main():
  userInput = input("Enter a WORD to be tested as a palindrome:")

  if (isPalindrome(userInput)):
    print(userInput + " is a palindrome!")
  else:
    print(userInput + " is NOT a palindrome!")

if __name__ == "__main__":
    main()
```

```
Enter a WORD to be tested as a palindrome:racecar
racecar is a palindrome!
```