Resumão

Comentários em Python são iniciados pelo caractere #, e se estendem até o final da linha física. Um comentário pode aparecer no início da linha, depois de um espaço em branco ou código, mas nunca dentro de uma string literal. O caractere # em uma string literal não passa de um caractere #. Uma vez que os comentários são usados apenas para explicar o código e não são interpretados pelo Python, eles podem ser omitidos ao digitar os exemplos.

In [1]:
```
# this is the first comment # Este é o primeiro comentario #Comentario no inicio da
spam = 1  # and this is the second comment # e este é o segundo comentario # Comenta
          # ... and now a third! # ... e agora um terceiro #comentário após um espaç
text = "# This is not a comment because it's inside quotes." # Jogo da velha dentro
```

No modo interativo, o valor da última expressão exibida é atribuída a variável _. Assim, ao utilizar Python como uma calculadora, fica mais fácil prosseguir com os cálculos, por exemplo:

In [2]:
```
8*4/2 #Atribui o valor do resultado desta operação a _
```

Out[2]:
```
16.0
```

In [3]:
```
Var= 4
```

In [4]:
```
# Esse _ é como o Ans de uma calculadora, guardando o resultado da ultima expressão
_*Var
```

Out[4]:
```
64.0
```

In [5]:
```
_
```

Out[5]:
```
64.0
```

Essa variável especial deve ser tratada como somente para leitura pelo usuário. Nunca lhe atribua explicitamente um valor — do contrário, estaria criando uma outra variável (homônima) independente, que mascararia a variável especial com seu comportamento mágico.

Além de int e float, o Python suporta outros tipos de números, tais como Decimal e Fraction. O Python também possui suporte nativo a números complexos, e usa os sufixos j ou J para indicar a parte imaginária (por exemplo, 3+5j).

Strings

Além de números, Python também pode manipular strings (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ('...') ou duplas ("...") e teremos o mesmo resultado 2. \ pode ser usada para escapar aspas:

In [6]:
```
'spam eggs'  # single quotes # aspas simples
```

Out[6]:
```
'spam eggs'
```

In [7]:
```
 "doesn't"  # ...or use double quotes instead # ...ou use aspas duplas
```

Out[7]:
```
"doesn't"
```

In [8]:
```
'"Yes," they said.'
```

Out[8]:
```
'"Yes," they said.'
```

A função print() produz uma saída mais legível, ao omitir as aspas e ao imprimir caracteres escapados e especiais:

In [9]:
```
'doesn\'t'  # use \' to escape the single quote...   # use \ 'para escapar das aspas
```

Out[9]:
```
"doesn't"
```

In [10]:
```
print('doesn\'t')
```

```
doesn't
```

In [11]:
```
"\"Yes,\" they said."
```

Out[11]:
```
'"Yes," they said.'
```

In [12]:
```
print("\"Yes,\" they said.")
```

```
"Yes," they said.
```

In [13]:
```
'"Isn\'t," they said.'
```

Out[13]:
```
'"Isn\'t," they said.'
```

In [14]:
```
print('"Isn\'t," they said.')
```

```
"Isn't," they said.
```

In [18]:
```
s = 'First line.\nSecond line.'  # \n means newline # \n significa nova linha
s  # without print(), \n is included in the output # sem print (), \ n é incluído na
```

Out[18]:
```
'First line.\nSecond line.'
```

In [19]:
```
print(s)  # with print(), \n produces a new line # com print (), \ n produz uma nova
```

```
First line.
Second line.
```

Se não quiseres que os caracteres sejam precedidos por \ para serem interpretados como caracteres especiais, poderás usar strings raw (N.d.T: “crua” ou sem processamento de caracteres de escape) adicionando um r antes da primeira aspa:

In [20]:
```
print('C:\some\name')  # here \n means newline!
```

```
C:\some
```

```
ame
```

In [21]:
```
print(r'C:\some\name')  # note the r before the quote
```

```
C:\some\name
```

As strings literais podem abranger várias linhas. Uma maneira é usar as aspas triplas: """..."" ou '''...'''. O fim das linhas é incluído automaticamente na string, mas é possível evitar isso adicionando uma \ no final. O seguinte exemplo:

In [22]:
```
print("""\
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
""")
```

```
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
```

Strings podem ser concatenadas (coladas) com o operador +, e repetidas com *:

In [23]:
```
# 3 times 'un', followed by 'ium'
3 * 'un' + 'ium'
```

Out[23]:
```
'unununium'
```

Duas ou mais strings literais (ou seja, entre aspas) ao lado da outra são automaticamente concatenados.

In [24]:
```
'Py' 'thon'
```

Out[24]:
```
'Python'
```

As strings podem ser indexadas (subscritas), com o primeiro caractere como índice 0. Não existe um tipo específico para caracteres; um caractere é simplesmente uma string cujo tamanho é 1. Índices também podem ser números negativos para iniciar a contagem pela direita. Note que dado que -0 é o mesmo que 0, índices negativos começam em -1.

In [28]:
```
word = 'Python'
```

In [29]:
```
word[0]
```

Out[29]:
```
'P'
```

In [30]:
```
word[5]
```

Out[30]:
```
'n'
```

In [31]:
```
word[-1]
```

Out[31]:
```
'n'
```

In [32]:
```
word[-2]
```

Out[32]:
```
'o'
```

In [33]:
```
word[-6]
```

Out[33]:
```
'P'
```

Além da indexação, o fatiamento também é permitido. Embora a indexação seja usada para obter caracteres individuais, fatiar permite que você obtenha substring:

In [34]:
```
word[0:2]  # caracteres da posição 0 (included) até a 2 (excluded)
```

Out[34]:
```
'Py'
```

In [35]:
```
word[2:5]  # caracteres da posição 0 (included) até a 5 (excluded)
```

Out[35]:
```
'tho'
```

In [37]:
```
word[:2]   # caracteres do inicio até a posição 2 (excluded)
```

Out[37]:
```
'Py'
```

In [38]:
```
word[4:]   # caracteres da posição 4 (included) até o final
```

Out[38]:
```
'on'
```

In [ ]:
```
word[-2:]  # caracteres do penúltimo (incluído) até o fim
```

Observe como o início sempre está incluído, e o fim sempre é excluído. Isso garante que s[:i] + s[i:] seja sempre igual a s:

In [39]:
```
word[:2] + word[2:]
```

Out[39]:
```
'Python'
```

In [40]:
```
word[:4] + word[4:]
```

Out[40]:
```
'Python'
```

Para índices positivos, o comprimento da fatia é a diferença entre os índices, se ambos estão dentro dos limites da string. Por exemplo, o comprimento de word[1:3] é 2.

A tentativa de usar um índice que seja muito grande resultará em um erro:word[42]. No entanto, os índices de fatiamento fora do alcance são tratados graciosamente (N.d.T: o termo original “gracefully” indica robustez no tratamento de erros) quando usados para fatiamento. Um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia:

In [41]:
```
word[4:42]
```

Out[41]:
```
'on'
```

In [42]:
```
word[42:]
```

Out[42]:
```
''
```

As strings do Python não podem ser alteradas — uma string é imutável. Portanto, atribuir a uma posição indexada na sequência resulta em um erro. Se você precisar de uma string diferente, deverá criar uma nova:

In [43]:
```
'J' + word[1:]
```

Out[43]:
```
'Jython'
```

In [44]:
```
word[:2] + 'py'
```

Out[44]:
```
'Pypy'
```

Listas

Atribuição a fatias também é possível, e isso pode até alterar o tamanho da lista ou remover todos os itens dela:

In [45]:
```
letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
letters
```

Out[45]:
```
['a', 'b', 'c', 'd', 'e', 'f', 'g']
```

In [46]:
```
# replace some values
letters[2:5] = ['C', 'D', 'E']
letters
```

Out[46]:
```
['a', 'b', 'C', 'D', 'E', 'f', 'g']
```

In [47]:
```
# now remove them
letters[2:5] = []
letters
```

Out[47]:
```
['a', 'b', 'f', 'g']
```

In [48]:
```
# clear the list by replacing all the elements with an empty list
letters[:] = []
letters
```

Out[48]:
```
[]
```

Capitulo 3

3.1 Condicionais - If/else/elif

O condicional If nos permite dizer ao computador para executar ações com base em um determinado conjunto de resultados.

```
                                                If significa se
```

Verbalmente podemos imaginar que estamos dizendo ao computador: "Ei, caso isso aconteça, execute esta ação".

A palavra reservada If não pode ser utilizada como nome de variável, pois o interpretador utiliza esta palavra reservada, para reconhecer o comando que ele deve executar. Após a palavra If vem a condição que não precisa estar entre parênteses e depois os dois pontos. Estes 2 pontos fazem parte da sintaxe assim como o recuo(a identação) que vem antes da ação que será executada caso a condição seja verdadeira.

Quando utiliza-se apenas o If testa-se apenas a condição verdadeira, quando esta é falsa não executa e passa para a próxima linha de codigo.

In [ ]:
```
# Condicional If # Se a condição 5>2 for verdadeira execute a operação print ("Pytho
if 5 > 2:
    print("Python funciona!")
```

O else quer dizer se não. Ele serve para controlar o que acontece quando a condição é falsa.

In [ ]:
```
# Statement If...Else
# Se a condição 5 < 2 for verdadeira execute esta ação #1, se não execute esta ação
if 5 < 2:
    print("Python funciona!") #1
else:
    print("Algo está errado!") #2
```

A condição após o If deve retornar um valor boleano (verdadeiro ou falso), podendo utilizar os operadores relacionais para testar se a condição é verdadeira ou falsa.

In [1]:
```
6 > 3
```

Out[1]:
```
True
```

In [2]:
```
3 > 7
```

Out[2]:
```
False
```

In [3]:
```
4 < 8
```

Out[3]:
```
True
```

In [4]:
```
4 >= 4
```

Out[4]:
```
True
```

In [5]:

```
#Operador de igualdade ==
if 5 == 5:     # Não pode ser apenas =, pois não pode ser atribuição. Esta condição
    print("Testando Python!")
```

```
Testando Python!
```

In [6]:

```
# Colocando diretamente a palavra True executa de imediato a operação(ação).
# Essa expressão é apenas para provar que a condição tem que ser/ter o resultado boo
if True:
    print('Parece que Python funciona!')
```

```
Parece que Python funciona!
```

In [7]:

```
# Atenção com a sintaxe
if 4 > 3            # Faltou os 2 pontos
    print("Tudo funciona!")
```

```
  File "C:\Users\carla\AppData\Local\Temp/ipykernel_6044/760903919.py", line 2
    if 4 > 3            # Faltou os 2 pontos
                        ^
SyntaxError: invalid syntax
```

Em tipográfia identação é o recuo de um texto em relação a sua margem. Na ciência da computação é um termo aplicado ao codigo fonte de um programa para ressaltar ou definir a estrutura de um algoritmo. No Python utiliza-se 1 tab ou 4 spaces(seja consistente use ou 1 ou outro, não misture). Na maioria das linguagens a identação é empregada com o objetivo de ressaltar a estrutura do algoritmo, aumentando a legitimidade do codigo, ou seja, nem toda a linguagem de programação exige a identação. Contudo, em Python utiliza-se a mudança de identação para definir a hierarquia entre blocos de códigos.

```
                                                A identação faz parte da
    sintaxe em python.
```

Em uma condicional if ela serve para que o interpretador saiba que ele deve executar este comando somente dentro do bloco if.

In [8]:

```
# Atenção com a sintaxe
if 4 > 3:
print("Tudo funciona!")     #Faltou o recuo. Errou por falta da identação.
```

```
  File "C:\Users\carla\AppData\Local\Temp/ipykernel_6044/3113919911.py", line 3
    print("Tudo funciona!")     #Faltou o recuo. Errou por falta da identação.
    ^
IndentationError: expected an indented block
```

Condicionais Aninhados

In [9]:

```
# Utilizando variável
idade = 18
if idade > 17:
    print("Você pode dirigir!")
```

```
Você pode dirigir!
```

Em algumas situações você pode querer testar uma condição, somente se uma outra condição tiver sido verdadeira. Neste caso pode-se utilizar um if aninhado. Uma instrução if dentro de

outra instrução if, que só será processada se a primeira instrução for verdadeira.

In [10]:
```
Nome = "Bob"      #Criou uma variavel
if idade > 13:        # Primeira checagem: Quer saber se a idade é maior que 13, par
    if Nome == "Bob":
        print("Ok Bob, você está autorizado a entrar!")
    else:  #Se não for Bob, executa esta ação
        print("Desculpe, mas você não pode entrar!")
```

```
Ok Bob, você está autorizado a entrar!
```

Contudo, não seria necessário fazer um if aninhado, pois este pode ser substituido por um operador lógico.

In [11]:
```
#Neste as duas condições tem que ser verdadeira, para executar a ação.
idade = 13
Nome = "Bob"
if idade >= 13 and Nome == "Bob":
    print("Ok Bob, você está autorizado a entrar!")
```

```
Ok Bob, você está autorizado a entrar!
```

In [12]:
```
# Neste caso basta que uma das condições seja verdadeira para executar a ação.
idade = 12
Nome = "Bob"
if (idade >= 13) or (Nome == "Bob"):
    print("Ok Bob, você está autorizado a entrar!")
```

```
Ok Bob, você está autorizado a entrar!
```

Elif

O Elif substitui a necessidade de criar várias estruturas de if...else aninhados.

In [13]:
```
dia = "Terça"
if dia == "Segunda":
    print("Hoje fará sol!")
else:
    print("Hoje vai chover!")
```

```
Hoje vai chover!
```

In [14]:
```
#Se a condição1 for true execute a ação1, se for falsa, verifique se a condição2 é t
if dia == "Segunda":
    print("Hoje fará sol!")
elif dia == "Terça":
    print("Hoje vai chover!")
else:
    print("Sem previsão do tempo para o dia selecionado")
```

```
Hoje vai chover!
```

In [15]:
```
if dia == "Segunda":
    print("Curso do trabalho")
elif dia == "Terça":
    print("Faxina")
elif dia == "Quarta":
    print("Estudar")
elif dia == "Quinta":
```

```
    print("Prova")
elif dia == "Sexta":
    print("Trabalhar")
elif dia == "Sabado":
    print("Compras")
else:
    print("Igreja")
```

```
Faxina
```

Operadores Lógicos

In [16]:

```
idade = 18
nome = "Bob"
if idade > 17:
    print("Você pode dirigir!")
```

```
Você pode dirigir!
```

In [17]:

```
#Tbm é possivel utilizar operadores lógicos, colocando-o entre duas condições
# O operador and chega que as duas condições são verdadeiras.
idade = 18
if idade > 17 and nome == "Bob":
    print("Autorizado!")
```

```
Autorizado!
```

Pode-se ainda fazer a leitura de valores a partir do prompt de comando. Solicitando que o usuario digite algum valor, através da função imput(), para então fazer a checagem.

In [19]:

```
# Usando mais de uma condição na cláusula if
# Chama a função imput e coloca uma mensagem para o usuario, este irá vai digitar al
disciplina = input('Digite o nome da disciplina: ')
nota_final = input('Digite a nota final (entre 0 e 100): ')

if disciplina == 'Geografia' and nota_final >= '70':    # A numeração esta entre '',
    print('Você foi aprovado!')
else:
    print('Lamento, acho que você precisa estudar mais!')

# Se não for geografia é necessário estudar mais.
```

```
Digite o nome da disciplina: História
Digite a nota final (entre 0 e 100): 100
Lamento, acho que você precisa estudar mais!
```

In [3]:

```
# Usando mais de uma condição na cláusula if e introduzindo Placeholders

disciplina = input('Digite o nome da disciplina: ')
nota_final = input('Digite a nota final (entre 0 e 100): ')
semestre = input('Digite o semestre (1 a 4): ')                              # O pyt

if disciplina == 'Geografia' and nota_final >= '50' and int(semestre) != 1:  #3 cond
    print('Você foi aprovado em %s com média final %r!' %(disciplina, nota_final)) #
else:
    print('Lamento, acho que você precisa estudar mais!')
```

```
Digite o nome da disciplina: Geografia
Digite a nota final (entre 0 e 100): 70
```

```
Digite o semestre (1 a 4): 3
Você foi aprovado em Geografia com média final '70'!
```

Os placeholders, são o %s e o %r, isso diz ao python que ele irá imprimir essa mensagem na tela, quando você encontrar o %s você vai pegar o primeiro valor da variável que está passado depois do %(, depois disso vai continuar imprimindo na tela até encontrar o proximo percenti %r, ai você vai pegar a segunda variavel que está após %(. O placeholder nos dá a chance de incluir na minha mensagem aquilo que o proprio usuario digitou durante a execução do programa.

3.2 Estruturas de repetição - Loop For

O loop for é uma das expressões mais comuns em qualquer linguagem de programação. Ele nos permite executar um comando ou um conjunto de comandos um número determinado de vezes. For significa para.

A forma geral do loop for é: Para cada item em uma série de itens execute este bloco e instruções, ou seja, execute estes comandos.

```
                                        O for valida cada item em uma
série de valores.

                                                For item in série-de-
items:
                                                        Executar
comandos.
```

O nome da variável item pode ser qualquer uma. Use o bom senso de escolher um nome que faça sentido e que você seja capaz de entender quando revisitar o seu codigo. Esta variável pode ser referenciada dentro de um loop, por exemplo, se você quiser utilizar o condicional if para executar algum tipo de verificação.

```
                                                For item in série-de-
items:
                                                        if item>0
                                                               Executar
comandos.
```

A série de itens neste caso é uma coleção e valores que pode ser qualquer sequencia de valores em Python, podendo utilizar o loop for em objetos sequenciais como:

1. Strings
2. Listas
3. Tuplas
4. Elementos de dicionários
5. Arquivos.

In [1]:
```
# Criando uma tupla e imprimindo cada um dos valores
tp = (2,3,4)
# Para cada item i dentro desta coleção(tupla=tp) imprima i. i é o nome do contador,
for i in tp:
    print(i)
```

```
2
3
4
```

In [2]:

```
# Criando uma lista e imprimindo cada um dos valores
ListaDoMercado = ["Leite", "Frutas", "Carne"]
# Para cada item na lista imprima o tem.
for i in ListaDoMercado:
    print(i)
```

```
Leite
Frutas
Carne
```

In [4]:

```
# Imprimindo os valores no intervalo entre 0 e 5 (exclusive)
# O range é outra estrutura de repetição. Ele cria uma sequencia com inicio no 1º el
for contador in range(0,5):
    print(contador)
```

```
0
1
2
3
4
```

In [6]:

```
# Imprimindo na tela os números pares da lista de números
# Também pode-se unir o loop com o condicional if.
lista = [1,2,3,4,5,6,7,8,9,10]
for num in lista:
    if num % 2 == 0:        # modulo (%) é o resto da divisão
        print (num)
# Divide-se cada elemento da lista por 2, se o resto for 0, imprima na tela.
```

```
2
4
6
8
10
```

In [7]:

```
# Listando os números no intervalo entre 0 e 101, com incremento em 2
#Cria-se um range de valores saltando 2 elementos.
for i in range(0,101,2):
    print(i)
```

```
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
```

```
34
36
38
40
42
44
46
48
50
52
54
56
58
60
62
64
66
68
70
72
74
76
78
80
82
84
86
88
90
92
94
96
98
100
```

In [8]:

```
# Strings também são sequências
#Para cada caracter na minha sequencia de caracteres imprime o caracter na tela.
for caracter in 'Python é uma linguagem de programação divertida!':
    print (caracter)
```

```
P
y
t
h
o
n

é

u
m
a

l
i
n
g
u
a
g
e
m
```

```
d
e

p
r
o
g
r
a
m
a
ç
ã
o

d
i
v
e
r
t
i
d
a
!
```

Loops Aninhados

In [12]:
```
# Loops aninhados
# Para cada item representado por i, dentro da minha sequencia de valores será execu
# Esta ação é uma outra operação de repetição(um outro loop), executando depois uma
for i in range(0,5):
    for a in range(1,3):
        print(a) # O que será imprimido é o a.
```

```
1
2
1
2
1
2
1
2
1
2
```

In [14]:
```
# Operando os valores de uma lista com loop for
listaB = [32,53,85,10,15,17,19]
soma = 0
#Vai percorrer a lista e para cada item da lista vai tamar 2 ações.
# 1º vai multiplicar cada elemento por 2.
# 2º vai somar a variavél soma com o resultado da multiplicação, depois vai atribuir
for i in listaB:
    double_i = i * 2
    soma += double_i

print(soma)
```

```
462
```

In [15]:
```
# Loops em lista de listas
#Também é possivel criar um loop for para percorrer uma lista de listas, que neste c
```

```
# Pode-se criar um loop for que vai percorrer cada valor do meu objeto e então impri
listas = [[1,2,3], [10,15,14], [10.1,8.7,2.3]]
for valor in listas:
    print(valor)
```

```
[1, 2, 3]
[10, 15, 14]
[10.1, 8.7, 2.3]
```

In [16]:

```
# Contando os itens de uma lista
lista = [5,6,10,13,17]
count = 0    # Em algumas situações você deve inicializar a sua variável.
# Para cada item da lista some um a variavel count, no final você terá a quantidade
for item in lista:
    count += 1

print(count)
```

```
5
```

In [20]:

```
# Contando o número de colunas
lst = [[1,2,3],[3,4,5],[5,6,7],[8,9,10]]
primeira_linha = lst[0] # Indexação. Puxou a primeiro elemento da lista, que é outra
count = 0
# Contasse os elementos da primeira linha, ou seja, a quantidade de colunas.
for column in primeira_linha:
    count = count + 1

print(count)
```

```
3
```

In [23]:

```
# Pesquisando em listas
# Vai percorrer a lista, e sempre que encontrar o número 5, irá imprimir a mensagem
listaC = [5, 6, 7, 10, 50]

# Loop através da lista
for item in listaC:
    if item == 5:
        print("Número encontrado na lista!")
```

```
Número encontrado na lista!
```

In [21]:

```
# Listando as chaves de um dicionário
dict = {'k1':'Python','k2':'R','k3':'Scala'}
# Para cada item do dicionario imprimir o item, neste caso, imprime as chaves do dic
for item in dict:
    print(item)
```

```
k1
k2
k3
```

In [22]:

```
# Imprimindo chave e valor do dicionário. Usando o método items() para retornar os i
for k,v in dict.items():
    print (k,v)
```

```
k1 Python
k2 R
k3 Scala
```

3.2 Estruturas de repetição - Loop While

O loop while em Python é uma das formas mais comuns de se executar iteração. A instrução while será executada repetidamente, seja uma unica instrução ou um grupo de instruções, desde que uma instrução seja verdadeira.

A palavra while significa enquanto em português.

```
                                    Valida cada item em uma série de
valores.

                                While(expressão1):
                                        print("comando executado caso
a expressão1 seja verdadeira")
```

Enquanto a expressão 1 for verdadeira execute estas instruções.

Obs.: Dentro do nosso codigo precisamos fazer com que a expressão1 deixe de ser verdadeira em algum momento, caso contrário, o loop ficará executando infinitamente e vai travar o seu computador. Uma forma de sair do loop infinito é parar a execução do comando com o stop, que tem como simbolo um quadrado preto na linha de ferramentas.

No loop for isso não acontece porque atribui-se uma quantidade de elementos(lista, dicionário, tupla, string) e quando eles acabam o loop é concluido.

In [1]:
```
# Usando o loop while para imprimir os valores de 0 a 9
counter = 0
while counter < 10:
    print(counter)
    counter = counter + 1  # Se não colocar essa linha de codigo, o counter sempre s
# Com o count =0 a condição sempre será verdadeira e o loop será infinito.
```

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

Enquanto a condição for verdadeira esse bloco de instruções vai ser executado. Em algum momento a condição deixará de ser verdadeira. Nesta hora entrará no bloco else e vai executar a sua instrução. O bloco else é uma forma de também controlar o que acontece com o loop while quando a condição for falsa.

In [2]:
```
# Também é possível usar a claúsula else para encerrar o loop while
x = 0

while x < 10:
    print ('O valor de x nesta iteração é: ', x)
    print (' x ainda é menor que 10, somando 1 a x')
    x += 1   # Vai somando um a x de modo que em algum momento ele deixará de ser me

else: #tbm pode-se utilizar a clausula else em while.
    print ('Loop concluído!')
```

```
O valor de x nesta iteração é:  0
 x ainda é menor que 10, somando 1 a x
O valor de x nesta iteração é:  1
 x ainda é menor que 10, somando 1 a x
O valor de x nesta iteração é:  2
 x ainda é menor que 10, somando 1 a x
O valor de x nesta iteração é:  3
 x ainda é menor que 10, somando 1 a x
O valor de x nesta iteração é:  4
 x ainda é menor que 10, somando 1 a x
O valor de x nesta iteração é:  5
 x ainda é menor que 10, somando 1 a x
O valor de x nesta iteração é:  6
 x ainda é menor que 10, somando 1 a x
O valor de x nesta iteração é:  7
 x ainda é menor que 10, somando 1 a x
O valor de x nesta iteração é:  8
 x ainda é menor que 10, somando 1 a x
O valor de x nesta iteração é:  9
 x ainda é menor que 10, somando 1 a x
Loop concluído!
```

Pass, Break, Continue

Tem-se algumas alternativas para interromper a execução de um loop While. Podemos usar as palavras reservadas pass, break e continue, para mudar o comportamento dentro do loop.

1. Break - frear/interromper a execução do loop.
2. Pass - Continuar seguindo a execução do loop dentro do while.
3. Continue - Pular uma iteração/condição/repetição.

In [3]:
```
counter = 0
while counter < 100:
    if counter == 4:  #Se o contador for =4, ele quer interromper o programa.
        break   # interrompe/freia o programa
    else: # Enquanto não tiver encontrado counter=4 utiliza-se o pass, no caso contu
        pass
    print(counter)
    counter = counter + 1
```

```
0
1
2
3
```

Pode-se utilizar o continue quando quiser pular uma iteração(repetição, condição, elemento). Para o verificador dentro da minha sequencia de elementos que é uma string. Se o verificador for igual a h, continue, ou seja, pula-se h e continua processando a nossa string, a nossa sequancia de elementos.

In [4]:
```
#Para cada caracter = verificador dentro da string Python. Se o verificador for =h,
for verificador in "Python":
    if verificador == "h":
        continue
    print(verificador)
```

```
P
y
t
```

```
o
n
```

While e For juntos

In [7]:

```
#original
for i in range(2,30):
    j = 2
    counter = 0
    while j < i:
        if i % j == 0:
            counter = 1
            j = j + 1
        else:
            j = j + 1

    if counter == 0:
        print(str(i) + " é um número primo")
        counter = 0
    else:
        counter = 0
```

```
2 é um número primo
3 é um número primo
5 é um número primo
7 é um número primo
11 é um número primo
13 é um número primo
17 é um número primo
19 é um número primo
23 é um número primo
29 é um número primo
```

In [10]:

```
#modificação1

for i in range(2,30):
    j = 2
    counter = 0 # Não precisa-se de outros counters=0, já que toda vez que mudar o v
                # será atribuido novamente a variável counter, pois counter=0 está n

    while j < i:        #Enquanto j<i, será feita a seguinte verificação:
        if i % j == 0:  #Se o resto é =0, counter será =1, caso contrario, nem entra
            counter = 1
        j = j + 1 # No original esta linha estava no if e no else, ou seja, ia acont
                  # resto ser igual a zero ou não, por isso, ele pode ficar no while

    if counter == 0:    # Se counter for =0, imprimirá na tela. Logo, se entrar no w
                        # serviu para excluir números que não são primos.
        print(str(i) + " é um número primo")
```

```
2 é um número primo
3 é um número primo
5 é um número primo
7 é um número primo
11 é um número primo
13 é um número primo
17 é um número primo
19 é um número primo
23 é um número primo
29 é um número primo
```

In [12]:

```
for i in range(2,30):
```

```
    j = 2
    #print("J:"+ str(j))
    counter = 0
    print("I:"+ str(i))
    print("J fora do while:"+str(j))
    while j < i:        #Enquanto j<i, será feita a seguinte verificação:
        if i % j == 0:  #Se o resto é =0, counter será =1 e somará 1 a j.
            print("entrou no if")
            counter = 1
        j = j + 1
        print("J dentro do while:"+ str(j))

    if counter == 0:    # Se counter for =0, imprimirá na tela
        print(str(i) + " é um número primo")
```

```
I:2
J fora do while:2
2 é um número primo
I:3
J fora do while:2
J dentro do while:3
3 é um número primo
I:4
J fora do while:2
entrou no if
J dentro do while:3
J dentro do while:4
I:5
J fora do while:2
J dentro do while:3
J dentro do while:4
J dentro do while:5
5 é um número primo
I:6
J fora do while:2
entrou no if
J dentro do while:3
entrou no if
J dentro do while:4
J dentro do while:5
J dentro do while:6
I:7
J fora do while:2
J dentro do while:3
J dentro do while:4
J dentro do while:5
J dentro do while:6
J dentro do while:7
7 é um número primo
I:8
J fora do while:2
entrou no if
J dentro do while:3
J dentro do while:4
entrou no if
J dentro do while:5
J dentro do while:6
J dentro do while:7
J dentro do while:8
I:9
J fora do while:2
J dentro do while:3
entrou no if
J dentro do while:4
```

```
J dentro do while:5
J dentro do while:6
J dentro do while:7
J dentro do while:8
J dentro do while:9
I:10
J fora do while:2
entrou no if
J dentro do while:3
J dentro do while:4
J dentro do while:5
entrou no if
J dentro do while:6
J dentro do while:7
J dentro do while:8
J dentro do while:9
J dentro do while:10
I:11
J fora do while:2
J dentro do while:3
J dentro do while:4
J dentro do while:5
J dentro do while:6
J dentro do while:7
J dentro do while:8
J dentro do while:9
J dentro do while:10
J dentro do while:11
11 é um número primo
I:12
J fora do while:2
entrou no if
J dentro do while:3
entrou no if
J dentro do while:4
entrou no if
J dentro do while:5
J dentro do while:6
entrou no if
J dentro do while:7
J dentro do while:8
J dentro do while:9
J dentro do while:10
J dentro do while:11
J dentro do while:12
I:13
J fora do while:2
J dentro do while:3
J dentro do while:4
J dentro do while:5
J dentro do while:6
J dentro do while:7
J dentro do while:8
J dentro do while:9
J dentro do while:10
J dentro do while:11
J dentro do while:12
J dentro do while:13
13 é um número primo
I:14
J fora do while:2
entrou no if
J dentro do while:3
J dentro do while:4
```

```
J dentro do while:5
J dentro do while:6
J dentro do while:7
entrou no if
J dentro do while:8
J dentro do while:9
J dentro do while:10
J dentro do while:11
J dentro do while:12
J dentro do while:13
J dentro do while:14
I:15
J fora do while:2
J dentro do while:3
entrou no if
J dentro do while:4
J dentro do while:5
entrou no if
J dentro do while:6
J dentro do while:7
J dentro do while:8
J dentro do while:9
J dentro do while:10
J dentro do while:11
J dentro do while:12
J dentro do while:13
J dentro do while:14
J dentro do while:15
I:16
J fora do while:2
entrou no if
J dentro do while:3
J dentro do while:4
entrou no if
J dentro do while:5
J dentro do while:6
J dentro do while:7
J dentro do while:8
entrou no if
J dentro do while:9
J dentro do while:10
J dentro do while:11
J dentro do while:12
J dentro do while:13
J dentro do while:14
J dentro do while:15
J dentro do while:16
I:17
J fora do while:2
J dentro do while:3
J dentro do while:4
J dentro do while:5
J dentro do while:6
J dentro do while:7
J dentro do while:8
J dentro do while:9
J dentro do while:10
J dentro do while:11
J dentro do while:12
J dentro do while:13
J dentro do while:14
J dentro do while:15
J dentro do while:16
J dentro do while:17
```

```
17 é um número primo
I:18
J fora do while:2
entrou no if
J dentro do while:3
entrou no if
J dentro do while:4
J dentro do while:5
J dentro do while:6
entrou no if
J dentro do while:7
J dentro do while:8
J dentro do while:9
entrou no if
J dentro do while:10
J dentro do while:11
J dentro do while:12
J dentro do while:13
J dentro do while:14
J dentro do while:15
J dentro do while:16
J dentro do while:17
J dentro do while:18
I:19
J fora do while:2
J dentro do while:3
J dentro do while:4
J dentro do while:5
J dentro do while:6
J dentro do while:7
J dentro do while:8
J dentro do while:9
J dentro do while:10
J dentro do while:11
J dentro do while:12
J dentro do while:13
J dentro do while:14
J dentro do while:15
J dentro do while:16
J dentro do while:17
J dentro do while:18
J dentro do while:19
19 é um número primo
I:20
J fora do while:2
entrou no if
J dentro do while:3
J dentro do while:4
entrou no if
J dentro do while:5
entrou no if
J dentro do while:6
J dentro do while:7
J dentro do while:8
J dentro do while:9
J dentro do while:10
entrou no if
J dentro do while:11
J dentro do while:12
J dentro do while:13
J dentro do while:14
J dentro do while:15
J dentro do while:16
J dentro do while:17
```

```
J dentro do while:18
J dentro do while:19
J dentro do while:20
I:21
J fora do while:2
J dentro do while:3
entrou no if
J dentro do while:4
J dentro do while:5
J dentro do while:6
J dentro do while:7
entrou no if
J dentro do while:8
J dentro do while:9
J dentro do while:10
J dentro do while:11
J dentro do while:12
J dentro do while:13
J dentro do while:14
J dentro do while:15
J dentro do while:16
J dentro do while:17
J dentro do while:18
J dentro do while:19
J dentro do while:20
J dentro do while:21
I:22
J fora do while:2
entrou no if
J dentro do while:3
J dentro do while:4
J dentro do while:5
J dentro do while:6
J dentro do while:7
J dentro do while:8
J dentro do while:9
J dentro do while:10
J dentro do while:11
entrou no if
J dentro do while:12
J dentro do while:13
J dentro do while:14
J dentro do while:15
J dentro do while:16
J dentro do while:17
J dentro do while:18
J dentro do while:19
J dentro do while:20
J dentro do while:21
J dentro do while:22
I:23
J fora do while:2
J dentro do while:3
J dentro do while:4
J dentro do while:5
J dentro do while:6
J dentro do while:7
J dentro do while:8
J dentro do while:9
J dentro do while:10
J dentro do while:11
J dentro do while:12
J dentro do while:13
J dentro do while:14
```

```
J dentro do while:15
J dentro do while:16
J dentro do while:17
J dentro do while:18
J dentro do while:19
J dentro do while:20
J dentro do while:21
J dentro do while:22
J dentro do while:23
23 é um número primo
I:24
J fora do while:2
entrou no if
J dentro do while:3
entrou no if
J dentro do while:4
entrou no if
J dentro do while:5
J dentro do while:6
entrou no if
J dentro do while:7
J dentro do while:8
entrou no if
J dentro do while:9
J dentro do while:10
J dentro do while:11
J dentro do while:12
entrou no if
J dentro do while:13
J dentro do while:14
J dentro do while:15
J dentro do while:16
J dentro do while:17
J dentro do while:18
J dentro do while:19
J dentro do while:20
J dentro do while:21
J dentro do while:22
J dentro do while:23
J dentro do while:24
I:25
J fora do while:2
J dentro do while:3
J dentro do while:4
J dentro do while:5
entrou no if
J dentro do while:6
J dentro do while:7
J dentro do while:8
J dentro do while:9
J dentro do while:10
J dentro do while:11
J dentro do while:12
J dentro do while:13
J dentro do while:14
J dentro do while:15
J dentro do while:16
J dentro do while:17
J dentro do while:18
J dentro do while:19
J dentro do while:20
J dentro do while:21
J dentro do while:22
J dentro do while:23
```

```
J dentro do while:24
J dentro do while:25
I:26
J fora do while:2
entrou no if
J dentro do while:3
J dentro do while:4
J dentro do while:5
J dentro do while:6
J dentro do while:7
J dentro do while:8
J dentro do while:9
J dentro do while:10
J dentro do while:11
J dentro do while:12
J dentro do while:13
entrou no if
J dentro do while:14
J dentro do while:15
J dentro do while:16
J dentro do while:17
J dentro do while:18
J dentro do while:19
J dentro do while:20
J dentro do while:21
J dentro do while:22
J dentro do while:23
J dentro do while:24
J dentro do while:25
J dentro do while:26
I:27
J fora do while:2
J dentro do while:3
entrou no if
J dentro do while:4
J dentro do while:5
J dentro do while:6
J dentro do while:7
J dentro do while:8
J dentro do while:9
entrou no if
J dentro do while:10
J dentro do while:11
J dentro do while:12
J dentro do while:13
J dentro do while:14
J dentro do while:15
J dentro do while:16
J dentro do while:17
J dentro do while:18
J dentro do while:19
J dentro do while:20
J dentro do while:21
J dentro do while:22
J dentro do while:23
J dentro do while:24
J dentro do while:25
J dentro do while:26
J dentro do while:27
I:28
J fora do while:2
entrou no if
J dentro do while:3
J dentro do while:4
```

```
entrou no if
J dentro do while:5
J dentro do while:6
J dentro do while:7
entrou no if
J dentro do while:8
J dentro do while:9
J dentro do while:10
J dentro do while:11
J dentro do while:12
J dentro do while:13
J dentro do while:14
entrou no if
J dentro do while:15
J dentro do while:16
J dentro do while:17
J dentro do while:18
J dentro do while:19
J dentro do while:20
J dentro do while:21
J dentro do while:22
J dentro do while:23
J dentro do while:24
J dentro do while:25
J dentro do while:26
J dentro do while:27
J dentro do while:28
I:29
J fora do while:2
J dentro do while:3
J dentro do while:4
J dentro do while:5
J dentro do while:6
J dentro do while:7
J dentro do while:8
J dentro do while:9
J dentro do while:10
J dentro do while:11
J dentro do while:12
J dentro do while:13
J dentro do while:14
J dentro do while:15
J dentro do while:16
J dentro do while:17
J dentro do while:18
J dentro do while:19
J dentro do while:20
J dentro do while:21
J dentro do while:22
J dentro do while:23
J dentro do while:24
J dentro do while:25
J dentro do while:26
J dentro do while:27
J dentro do while:28
J dentro do while:29
29 é um número primo
```

3.3 Estruturas de repetição - Range

Os loops for e while são as principais estruturas de repetição em Python. Entretando, em algumas situações, nos queremos repetir um determinado conjunto de elementos um número

determinado de vezes, para isso podemos utilizar a função range que cria um range de valores. Essa função não é exatamente uma estrutura de repetição como loop/while, mas nos permite repetir o conjunto de elementos um número determinado de vezes.

A função range() nos permite criar uma lista de números em um intervalo especifico. A função range() tem o seguinte formato:

```
range([start],[stop],
[step])
```

1. Start - número que inicia a sequência;
2. Stop - número que encerra a sequência(não é incluido na sequência);
3. Step - diferença entre cada número da sequência.

O caso abaixo, vai retornar uma lista de números começando com o valor 50; A lista vai até o valor 101, sem inclui-lo; E os números serão trazidos de 2 em 2. Ou seja, serão trazidos os números pares, começando por 50 e terminando em 100. Desta forma, foi criada uma sequencia(uma repetição), de valores seguindo uma ordem especifica e uma quantidade determinada.

In [7]:
```
range(50,101,2)
```

Out[7]:
```
range(50, 101, 2)
```

In [8]:
```
# Imprimindo números pares entre 50 e 101
for i in range(50, 101, 2): # Para cada item(i) dentro da sequencia tal(range(50,101
    print(i)
```

```
50
52
54
56
58
60
62
64
66
68
70
72
74
76
78
80
82
84
86
88
90
92
94
96
98
100
```

In [9]:
```
#Como esta omitindo o step, vai considerar 1 como padrão.
```

```
for i in range(3, 6):
    print (i)
```

```
3
4
5
```

Pode-se também criar uma lista inversa, em vez de andar para frente, anda-se para trás, colocando valores negativos. Pense em uma reta onde o centro é zero, na esquerda estão os valores negativos e na direita os valores positivos.

In [11]:
```
# Começa com 0; Foi até 20; Saltando -2.
for i in range(0, -20, -2):
    print(i)
```

```
0
-2
-4
-6
-8
-10
-12
-14
-16
-18
```

In [14]:
```
# Isso é util quando se tem uma sequencia que não possui tamanho fixo, deixando mais
lista = ['Morango', 'Banana', 'Abacaxi', 'Uva'] #cria-se uma lista
lista_tamanho = len(lista) #verifica-se seu tamanho
for i in range(0, lista_tamanho):    #Adiciona-se o tamanho no Stop e 0 no Start, pa
    print(lista[i]) # utiliza-se o indice para puxar cada valor da lista  no for e i
#    print(i)        # pois se colocasse apenas o i daria o indice, ou seja, um núme
```

```
Morango
Banana
Abacaxi
Uva
```

In [15]:
```
# Tudo em Python é um objeto
type(range(0,3))
```

Out[15]:
```
range
```

Resumão

Provavelmente o mais conhecido comando de controle de fluxo é o if .

Pode haver zero ou mais partes elif, e a parte else é opcional. A palavra-chave 'elif' é uma abreviação para 'else if', e é útil para evitar indentação excessiva. Uma sequência if ... elif ... elif ... substitui os comandos switch ou case, encontrados em outras linguagens.

O comando for em Python é um pouco diferente do que costuma ser em C ou Pascal. Ao invés de sempre iterar sobre uma progressão aritmética de números (como no Pascal), ou permitir ao usuário definir o passo de iteração e a condição de parada (como C), o comando for do Python itera sobre os itens de qualquer sequência (seja uma lista ou uma string), na ordem que aparecem na sequência. Por exemplo:

In [3]:

```
# Measure some strings:
words = ['cat', 'window', 'defenestrate']
for w in words:
    print(w, len(w)) # len() é comprimento, por isso, contou os caracteres de cada e
```

```
cat 3
window 6
defenestrate 12
```

Se você precisa iterar sobre sequências numéricas, a função embutida range() é a resposta. Ela gera progressões aritméticas:

In [6]:
```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

O ponto de parada fornecido nunca é incluído na lista; range(10) gera uma lista com 10 valores, exatamente os índices válidos para uma sequência de comprimento 10. É possível iniciar o intervalo com outro número, ou alterar a razão da progressão (inclusive com passo negativo):

range(5, 10)

```
5, 6, 7, 8, 9
```

range(0, 10, 3)

```
0, 3, 6, 9
```

range(-10, -100, -30)

```
-10, -40, -70
```

Para iterar sobre os índices de uma sequência, combine range() e len() da seguinte forma:

In [7]:
```
a = ['Mary', 'had', 'a', 'little', 'lamb']
for i in range(len(a)): # Len é o comprimento da lista, neste caso 5.
    print(i, a[i])
```

```
0 Mary
1 had
2 a
3 little
4 lamb
```

Na maioria dos casos, porém, é mais conveniente usar a função enumerate().

Uma coisa estranha acontece se você imprime um intervalo:

In [8]:
```
print(range(10))
```

```
range(0, 10)
```

Em muitos aspectos, o objeto retornado pela função range() se comporta como se fosse uma lista, mas na verdade não é. É um objeto que retorna os itens sucessivos da sequência desejada quando você itera sobre a mesma, mas na verdade ele não gera a lista, economizando espaço.

Dizemos que um objeto é iterável, isso é, candidato a ser alvo de uma função ou construção que espera alguma coisa capaz de retornar sucessivamente seus elementos um de cada vez. Nós vimos que o comando for é um exemplo de construção, enquanto que um exemplo de função que recebe um iterável é sum():

In [9]:
```
sum(range(4))  # 0 + 1 + 2 + 3
```

Out[9]:
```
6
```

Mais tarde, veremos mais funções que retornam e recebem iteráveis como argumentos. Por último, você deve estar curioso sobre como pegar uma lista de um intervalo. Aqui está a solução:

In [ ]:
```
#Para pegar uma lista de um intervalo, transforme-o em lista, através da função list
list(range(4))
```

O comando break , como no C, sai imediatamente do laço de repetição mais interno, seja for ou while.

Laços podem ter uma cláusula else, que é executada sempre que o laço se encerra por exaustão do iterável (no caso do for) ou quando a condição se torna falsa (no caso do while), mas nunca quando o laço é interrompido por um break. Isto é exemplificado no próximo exemplo que procura números primos:

In [10]:
```
for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print(n, 'equals', x, '*', n//x)
            break
    else:
        # loop fell through without finding a factor
        print(n, 'is a prime number')
```

```
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

In [11]:
```
list(range(2,2))
```

Out[11]:
```
[]
```

In [12]:
```
list(range(2,3))
```

Out[12]:
```
[2]
```

(Sim, o código está correto. Olhe atentamente: a cláusula else pertence ao laço for, e não ao comando if.)

Quando usado em um laço, a cláusula else tem mais em comum com a cláusula else de um comando try do que com a de um comando if: a cláusula else de um comando try executa quando não ocorre exceção, e o else de um laço executa quando não ocorre um break. Para mais informações sobre comando try e exceções, veja Tratamento de exceções.

A instrução continue , também emprestada da linguagem C, continua com a próxima iteração do laço:

In [13]:
```
for num in range(2, 10):
    if num % 2 == 0:
        print("Found an even number", num)
        continue
    print("Found an odd number", num)
```

```
Found an even number 2
Found an odd number 3
Found an even number 4
Found an odd number 5
Found an even number 6
Found an odd number 7
Found an even number 8
Found an odd number 9
```

O comando pass não faz nada. Pode ser usada quando a sintaxe exige um comando mas a semântica do programa não requer nenhuma ação. Por exemplo:

In [ ]:
```
while True:
    pass  # Busy-wait for keyboard interrupt (Ctrl+C)
```

Isto é usado muitas vezes para se definir classes mínimas:

In [15]:
```
class MyEmptyClass:
    pass
```

Outra ocasião em que o pass pode ser usado é como um substituto temporário para uma função ou bloco condicional, quando se está trabalhando com código novo, ainda indefinido, permitindo que mantenha-se o pensamento num nível mais abstrato. O pass é silenciosamente ignorado:

In [16]:
```
def initlog(*args):
    pass   # Remember to implement this!
```

3.4 Métodos

Nós já vimos alguns exemplos de métodos quando aprendemos sobre estruturas de dados em Python. Os metódos são essencialmente funções incorporadas em objetos, por isso a sua criação utiliza proogramação orientada a objeto(OOP) e classes.

Métodos permitem executar ações especificas no objeto e podem também ter argumentos, exatamente como uma função.

Os métodos são executados sob a forma:

```
objeto.metodo(arg1,arg2,etc...)
```

Com o jupyter Notebook podemos ver rapidamente todos os métodos possiveis para o objeto usando a tecla TAB. Por exemplo, os métodos para o objeto lista são:

```
                        .append         .pop            .extend
.reverse

                        .count          .remove         .insert
.sort
```

Nomedalista.teclaTAB -> Mostra a lista de métodos, que pode variar de acordo com o objeto (lista, tupla, dicionario, string, ...). Métodos que funcionam em uma lista podem ser diferentes, dos métodos que funcionam em uma tupla.

In [1]:
```
# Criando uma lista
lst = [100, -2, 12, 65, 0]
```

In [2]:
```
# Usando um método do objeto lista
lst.append(10)
```

In [21]:
```
# Imprimindo a lista
lst
```

Out[21]:
```
[100, -2, 12, 65, 0, 10]
```

In [22]:
```
# Usando um método do objeto lista
lst.count(10)
```

Out[22]:
```
1
```

Se quiser saber o que cada um dos métodos faz, pode-se usar a função help():

In [23]:
```
# A função help() explica como utilizar cada método de um objeto
help(lst.count)
```

```
Help on built-in function count:

count(value, /) method of builtins.list instance
    Return number of occurrences of value.
```

Pode-se também listar todos os metódos e atributos do objeto utilizando a função dir():

In [24]:
```
# A função dir() mostra todos os métodos e atributos de um objeto
dir(lst)
```

Out[24]:
```
['__add__',
 '__class__',
 '__class_getitem__',
 '__contains__',
 '__delattr__',
 '__delitem__',
```

```
'__dir__',
'__doc__',
'__eq__',
'__format__',
'__ge__',
'__getattribute__',
'__getitem__',
'__gt__',
'__hash__',
'__iadd__',
'__imul__',
'__init__',
'__init_subclass__',
'__iter__',
'__le__',
'__len__',
'__lt__',
'__mul__',
'__ne__',
'__new__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__reversed__',
'__rmul__',
'__setattr__',
'__setitem__',
'__sizeof__',
'__str__',
'__subclasshook__',
'append',
'clear',
'copy',
'count',
'extend',
'index',
'insert',
'pop',
'remove',
'reverse',
'sort']
```

In [25]:
```
a = 'Isso é uma string'
```

In [26]:
```
# O método de um objeto pode ser chamado dentro de uma função, como print()
print (a.split())
```

```
['Isso', 'é', 'uma', 'string']
```

3.4 Funções

As funções também conhecidas por subrotinas são muito utilizadas em programação. Um dos grandes beneficios é não precisar copiar o codigo todas as vezes que precisar executar uma determinada operação. Além de deixar a leitura do codigo muito mais intuitiva.

Função é um dispositivo que agrupa um conjunto de instruções para que elas possam ser executadas mais de uma vez. Funções também permitem especificar os parâmetros que podem servir como entrada para as funções.

Em um nivel mais fundamental a construção de funções nos permite reutilizar codigo, sem ter

que escrevê-lo novamente. Nas aulas de string utilizamos a função len() para obter o comprimento de uma string. Com as funções escrevemos o codigo uma vez e repetimos a mesma instrução, fazendo a chamada a função, quantas vezes forem necessárias.

Utilizamos a palavra def seguido do nome da função e os argumentos se necessários. Na sequência pode-se ter comentários codigos e por último tem-se o retorno desejado da função.

O formato geral de uma função é:

```
def nomeDaFunção(arg1,arg2):
    '''
    Aqui vão os comentários,
    documentando a função.
    '''
    <Aqui vai o seu código>
    <Retorno desejado pela função>
```

Funções em Python são a forma de escrever a sua lógica em um único pacote e usa-la em diferentes lugares no seu código e quantas vezes quiser.

Uma função recebe como entrada: Argumentos, variáveis globais e arquivos/Streams de dados.

Uma função pode conter variáveis locais, usadas apenas internamente, dentro da função. O codigo da função, então, será executado e vai gerar uma ou mais saídas que pode ser um resultado especifico, variáveis globais, arquivos e etc de acordo com a nossa necessidade.

Quando executa a definição de uma função, tem-se na memoria do computador uma função criada em PYthon. Agora pode-se usa-la através de uma chamada desta função. Contudo ao criar outra função com mesmo nome, sobescreve(como uma fita casséte), ou seja, a que foi criada primeiro deixa de existir na memoria do computador.

Uma dica para o nome da função: Se for duas palavras escreve as duas concatenadas, onde a primeira letra da primeira palavra é minuscula e a primeira letra da segunda palavra é maiuscula.

In [1]:
```
# Definindo uma função
def primeiraFunc():
    print('Hello World')
```

In [2]:
```
#Chama a função
primeiraFunc()
```

```
Hello World
```

In [3]:
```
# Definindo uma função com parâmetro
def primeiraFunc(nome):   #Quando a função recebe o parâmetro, ela vai passar o valo
    print('Hello %s' %(nome)) #Utilizando placeholders
```

In [4]:
```
primeiraFunc('Aluno')
```

```
Hello Aluno
```

In [15]:
```
def funcLeitura():
    for i in range(0, 5):
        print("Número " + str(i)) #Tem que colocar o str, para transformar em string
```

In [16]:
```
funcLeitura()
```

```
Número 0
Número 1
Número 2
Número 3
Número 4
```

In [17]:
```
# Função para somar números
def addNum(firstnum, secondnum):
    print("Primeiro número: " + str(firstnum))
    print("Segundo número: " + str(secondnum))
    print("Soma: ", firstnum + secondnum)
```

In [18]:
```
# Chamando a função e passando parâmetros
addNum(45, 3)
```

```
Primeiro número: 45
Segundo número: 3
Soma:  48
```

Variáveis locais e globais

In [6]:
```
# Variável Global
var_global = 10  # Esta é uma variável global

def multiply(num1, num2):
    var_global = num1 * num2  # Esta é uma variável local(existe apenas dentro da fu
    print(var_global) #Retorno desejado pela função
```

In [7]:
```
multiply(5, 25)
```

```
125
```

In [8]:
```
#O que será impresso é o valor da global, já que fora da função não se tem controle
print(var_global)
```

```
10
```

In [9]:
```
# De maneira mais explicita temos:
# Variável Local
var_global = 10  # Esta é uma variável global
def multiply(num1, num2):
    var_local = num1 * num2   # Esta é uma variável local
    print(var_local)
```

In [10]:
```
multiply(5, 25)
```

```
125
```

In [11]:
```
print(var_local) #A variável local só existe dentro da função.
```

```
---------------------------------------------------------------------------
```

```
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_348/3637017460.py in <module>
----> 1 print(var_local) #A variável local só existe dentro da função.

NameError: name 'var_local' is not defined
```

Funções Built-in

In [12]:
```
#Retorna o valor absoluto(positivo)
abs(-56)
```

Out[12]:
```
56
```

In [13]:
```
abs(23)
```

Out[13]:
```
23
```

In [14]:
```
#Retorna o valor booleano(verdadeiro ou falso)
bool(0)
```

Out[14]:
```
False
```

In [15]:
```
bool(1)
```

Out[15]:
```
True
```

Funções str, int, float

In [16]:
```
# Erro ao executar por causa da conversão, pois tem uma diferença de tipo de dados
idade = input("Digite sua idade: ") # O input recebe uma string. A resposta vem em f
if idade > 13: #Comparou uma string com valor número
    print("Você pode acessar o Facebook")
```

```
Digite sua idade: 14
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_348/2063637719.py in <module>
      1 # Erro ao executar por causa da conversão, pois tem uma diferença de tipo de
dados
      2 idade = input("Digite sua idade: ") # O input recebe uma string. A resposta
 vem em formato string
----> 3 if idade > 13: #Comparou uma string com valor número
      4     print("Você pode acessar o Facebook")

TypeError: '>' not supported between instances of 'str' and 'int'
```

In [17]:
```
# Pode-se corrigir isso usando a função int para converter o valor digitado
idade = int(input("Digite sua idade: "))
if idade > 13:
    print("Você pode acessar o Facebook")
```

```
Digite sua idade: 14
Você pode acessar o Facebook
```

In [18]:
```
int("26") #Converte string para valor número
```

Out[18]:
```
26
```

In [19]:
```
float("123.345") #Converte string para valor númerico
```

Out[19]:
```
123.345
```

In [20]:
```
str(14) #Converte o valor noumerico para string
```

Out[20]:
```
'14'
```

In [21]:
```
len([23,34,45,46]) #Verifica o comprimento de um objeto
```

Out[21]:
```
4
```

In [28]:
```
array = ['a', 'b', 'c']
array
```

Out[28]:
```
['a', 'b', 'c']
```

In [29]:
```
max(array)
```

Out[29]:
```
'c'
```

In [30]:
```
min(array)
```

Out[30]:
```
'a'
```

In [31]:
```
array = ['a', 'b', 'c', 'd', 'A', 'B', 'C', 'D']
```

In [32]:
```
array
```

Out[32]:
```
['a', 'b', 'c', 'd', 'A', 'B', 'C', 'D']
```

In [33]:
```
max(array)
```

Out[33]:
```
'd'
```

In [34]:
```
min(array) #Por causa da codificação o valor A maiusculo é o valor minimo.
```

Out[34]:
```
'A'
```

In [35]:
```
list1 = [23, 23, 34, 45]
```

In [36]:
```
sum(list1) #Soma os valores da lista
```

Out[36]:
```
125
```

Criando funções usando outras funções

In [37]:
```
import math #Importou esse pacote(é um pacote de matemática). Um pacote possui um co

def numPrimo(num):
    '''
    Verificando se um número
    é primo.
    '''
    if (num % 2) == 0 and num > 2:
        return "Este número não é primo"
    for i in range(3, int(math.sqrt(num)) + 1, 2):
        if (num % i) == 0:
            return "Este número não é primo"
    return "Este número é primo"
```

In [43]:
```
numPrimo(541)
```

Out[43]:
```
'Este número é primo'
```

In [42]:
```
list(range(3, int(math.sqrt(541)) + 1, 2)) #não entrou no if dentro do for
```

Out[42]:
```
[3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23]
```

In [46]:
```
numPrimo(9)
```

Out[46]:
```
'Este número não é primo'
```

Fazendo split dos dados

In [60]:
```
# Fazendo split dos dados #Com uma função dentro de outra função (___.split())
def split_string(text):
    return text.split(" ")  #Divide o texto por espaços.
```

In [61]:
```
texto = "Esta função será bastante útil para separar grandes volumes de dados."
```

In [62]:
```
split_string(texto)
```

Out[62]:
```
['Esta',
 'função',
 'será',
 'bastante',
 'útil',
 'para',
 'separar',
 'grandes',
 'volumes',
 'de',
 'dados.']
```

In [65]:
```
# Podemos atribuir o output de uma função, para uma variável #Pode salvar o resultad
token = split_string(texto)
```

In [66]:
```
token
```

Out[66]:
```
['Esta',
 'função',
 'será',
 'bastante',
 'útil',
 'para',
 'separar',
 'grandes',
 'volumes',
 'de',
 'dados.']
```

In [64]:
```
# Isso divide a string em uma lista.
print(split_string(texto))
```

```
['Esta', 'função', 'será', 'bastante', 'útil', 'para', 'separar', 'grandes', 'volume
s', 'de', 'dados.']
```

In [53]:
```
caixa_baixa = "Este Texto Deveria Estar Todo Em LowerCase"
```

In [54]:
```
#Com uma função dentro de outra função (___.lower())
def lowercase(text):
    return text.lower() #Coverte tudo para minusculo
```

In [57]:
```
lowercase(caixa_baixa)
```

Out[57]:
```
'este texto deveria estar todo em lowercase'
```

In [67]:
```
# Funções com número variável de argumentos #Quando não se sabe quantos parâmetros i
#A decisão de nº de arg será em tempo de execução, ao invés de determinar um unico a
# O * indica um conjunto
def printVarInfo( arg1, *vartuple ):
   # Imprimindo o valor do primeiro argumento
    print ("O parâmetro passado foi: ", arg1)

   # Imprimindo o valor do segundo argumento
    for item in vartuple:
        print ("O parâmetro passado foi: ", item)
    return;
```

In [68]:
```
# Fazendo chamada à função usando apenas 1 argumento
printVarInfo(10)
```

```
O parâmetro passado foi:  10
```

In [69]:
```
printVarInfo('Chocolate', 'Morango', 'Banana')
```

```
O parâmetro passado foi:  Chocolate
O parâmetro passado foi:  Morango
O parâmetro passado foi:  Banana
```

3.5 Expressões Lambda

Python nos oferece um tipo especial de função que é muito flexivel e muito util em diversas situações que é conhecido como função in-line, função anônima ou apenas expressão lambda.

Expressões lambda nos permite criar funções "anônimas". Isso significa que podemos fazer rapidamente funções ad-hoc(criadas em tempo de execução) sem a necessidade de definir uma função usando a palavra reservada def.

Objetos de função desenvolvidos executando expressões lambda funcionam exatamente da mesma forma como aqueles criados e atribuidos pela palavra reservada def. Mas há algumas diferenças fundamentais que fazem lambda útil em funções especializadas:

1. O corpo do lambda é uma única expressão, não um bloco de instruções.
2. O corpo do lambda é semelhante a uma instrução de retorno do corpo def.

Expressões lambda realmente são uteis, quando usadas em conjunto com as funções map(), filter() e reduce().

Expressões lambda são usadas para criar funções simples. Como:

```
lambda x: x**2
```

Diferença entre def e lambda para criar funções:

def -> cria um objeto e atribui um nome a ele(nome da função).

lambda -> cria um objeto, mas retorna como resultado em tempo de execução.

In [70]:
```
# Definindo uma função - 3 linhas de código
def potencia(num):
    result = num**2 #bloco de instruções
    return result
```

In [71]:
```
potencia(5)
```

Out[71]: 25

In [72]:
```
# Definindo uma função - 2 linhas de código
def potencia(num):
    return num**2 #colocando as instruções diretamente no returno
```

In [73]:
```
potencia(5)
```

Out[73]: 25

In [74]:
```
# Definindo uma função - 1 linha de código
def potencia(num): return num**2
```

In [75]:
```
potencia(5)
```

Out[75]: 25

In [76]:
```
# Definindo uma expressão lambda
potencia = lambda num: num**2 #A expr. lambda ñ tem nome de função, atribui-se a uma
```

In [77]:
```
potencia(5)
```

Out[77]:
```
25
```

In [87]:
```
# Lembre: operadores de comparação retornam boolean, true or false
Par = lambda x: x%2==0
```

In [88]:
```
Par(3)
```

Out[88]:
```
False
```

In [89]:
```
Par(4)
```

Out[89]:
```
True
```

In [90]:
```
first = lambda s: s[0]
```

In [91]:
```
first('Python')
```

Out[91]:
```
'P'
```

In [92]:
```
reverso = lambda s: s[::-1]
```

In [93]:
```
reverso('Python')
```

Out[93]:
```
'nohtyP'
```

In [ ]:
```
addNum = lambda x,y : x+y
```

In [ ]:
```
addNum(2,3)
```

Resumão

In [ ]:
```
# Fibonacci series:
# the sum of two elements defines the next
a, b = 0, 1
print(a)
print(b)
```

In [ ]:
```
while a < 10:
    print(a)
    a, b = b, a+b
```