A História do Python

Python é uma das linguagens de programação mais populares em todo o mundo. Criada em 1991 por Guido van Rossum [1], um programador holandês, a linguagem foi desenvolvida com o objetivo de ser fácil de aprender e de usar. Apesar de simples, é uma linguagem muito poderosa, que pode ser usada para desenvolver e administrar grandes sistemas.

A origem do nome Python vem do programa de televisão Monty Python's Flying Circus, que era popular entre os programadores da época. Guido van Rossum queria um nome que fosse curto, único e engraçado, e Python acabou sendo a escolha perfeita.

Nos anos seguintes, a popularidade do Python cresceu rapidamente, principalmente entre os desenvolvedores que precisavam de uma linguagem simples e fácil de usar para criar scripts e programas de automação.

Dentre as características existentes nesta linguagem, destaca-se a simplicidade na escrita de códigos o que facilita o aprendizado da programação, aumenta a legibilidade, sendo visualmente mais limpa e de fácil compreensão e leitura. Isto acontece por não haver o uso excessivo de pontuações, chaves, parênteses ou colchetes no código. Também possui uma portabilidade muito grande para diversas plataformas, além de ser possível utilizar trechos de códigos em outras linguagens.

É um software livre, ou seja, permite que usuários e colaboradores possam modificar seu código fonte e compartilhar essas novas atualizações, contribuindo para o constante aperfeiçoamento da linguagem. A especificação da linguagem é mantida pela empresa Python Software Foundation (PSF) [2].

Com o tempo, o Python se tornou uma das linguagens mais populares em diversas áreas, incluindo desenvolvimento web, ciência de dados, inteligência artificial e aprendizado de máquina.

Uma das razões para a popularidade do Python é a sua comunidade de desenvolvedores ativa e engajada. Essa comunidade desenvolveu uma vasta biblioteca de módulos e pacotes que podem ser usados para uma ampla variedade de tarefas, desde a criação de gráficos até a análise de dados e a construção de aplicativos web.

Atualmente, o Python é uma das linguagens de programação mais valorizadas no mercado de trabalho, onde empresas de tecnologia como Google, Amazon, Microsoft e Facebook a utilizam, assim como muitos profissionais de tecnologia estão se dedicando a linguagem como uma opção de carreira lucrativa e envolvente.

Se você é um autodidata ávido por colocar a "mão na massa" e não quer acompanhar todo o material disponível aqui, siga direto para a documentação da linguagem [3] e boa diversão! Ah! Não esqueça de ler o apêndice sobre a instalação de Python em seu computador para tornar a jornada ainda mais proveitosa.

Um pouco sobre o criador do Python

Na apresentação sobre sua história de vida [4], Guido deixa algumas reflexões que considero importantes e compartilho alguns trechos a seguir:

```
"Release early, release often"
```

...que podemos interpretar como a habilidade de desenvolver rapidamente e com frequência.

Se refletirmos sobre essa possibilidade disponível imediatamente com o uso da linguagem e do GitHub, isso faz todo sentido.

```
Normalmente, quando você pede a um programador para explicar a um leigo o
que é uma linguagem de programação, ele dirá que é como você diz a um
computador o que fazer. Mas se isso fosse tudo, por que eles seriam tão
apaixonados por linguagens de programação quando conversam entre si?

Na realidade, as linguagens de programação são como os programadores
expressam e comunicam ideias — e o público para essas ideias são outros
programadores, não computadores. O motivo: o computador pode cuidar de si
mesmo, mas os programadores estão sempre trabalhando com outros
programadores, e ideias mal comunicadas podem causar fracassos caros. Na
verdade, as ideias expressas em uma linguagem de programação também
costumam chegar aos usuários finais do programa — pessoas que nunca lerão
ou mesmo saberão sobre o programa, mas que, no entanto, são afetadas por
ele.

História real: a primeira versão do Google foi escrita em Python. O motivo:
Python era a linguagem certa para expressar as ideias originais que Larry
Page e Sergey Brin tiveram sobre como indexar a web e organizar os
resultados de pesquisa. E eles também poderiam executar suas ideias em um
computador!
```

Em suma, o entusiamo sobre o que você é capaz de criar e compartilhar com outras pessoas foi e é o combustível que anima Guido em sua jornada e muito provavelmente aquilo que é percebido por todos aqueles que apreciam a programação e as possibilidades que profissão nos entrega.

Variáveis

Variáveis são espaços em memória, utilizados para armazenar e manipular dados. Em Python, os tipos de dados básicos são: tipo inteiro (números negativos e positivos sem casas decimais), tipo float (números negativos e positivos com casas decimais), tipo string (composto por um conjunto de caracteres) e tipo booleano (cujos valores lógicos são verdadeiro ou falso). Cada variável pode armazenar apenas um tipo de dado a cada instante. Diferentemente de outras linguagens de programação, não é preciso declarar de que tipo que será usado para cada variável no início do programa. Quando se faz uma atribuição de valor, automaticamente a variável se torna do tipo do valor armazenado.

Exemplo:

```
Inteiros (int): Números inteiros, como 5, -3, 42.

Ponto flutuante (float): Números decimais, como 3.14, -0.001, 2.0.

Strings (str): Sequências de caracteres, como "Olá, mundo!", 'Python'.
```

```
Booleanos (bool): Valores lógicos que podem ser True ou False.
```

Além desses, existem outros tipos de dados mais complexos, como listas, tuplas, dicionários e conjuntos.

Podemos realizar a atribuição direta para a variável da seguinte forma:

```
a = 10          # onde 'a' se torna uma variável do tipo inteiro

b = 3.14        # onde 'b' se torna uma variável do tipo float

c = 'Olá Mundo' # onde 'c' se torna uma variável do tipo string

d = True        # onde 'd' se torna uma variável do tipo bool  (True/False)
```

A atribuição de valor para uma variável pode ser feita utilizando o comando input(), que solicita ao usuário o valor a ser atribuído.

```
nome = input("Entre com seu nome: ")
```

O comando input(), sempre vai retornar uma string. Nesse caso, para retornar dados do tipo inteiro ou float, é preciso converter o tipo do valor lido. Para isso, utiliza-se o int (string) para converter para o tipo inteiro, ou float (string) para converter para o tipo float.

```
idade = int(input("Entre com sua idade: "))

altura = float(input("Entre com sua altura: "))
```

Em Python, os nomes das variáveis devem ser iniciados com uma letra, mas podem possuir outros tipos de caracteres, como números e símbolos. O traço abaixo _ também é aceito no início de nomes de variáveis ou no lugar de um espaço entre palavras.

```
velocidade90

_salario

salario_bruto
```

Strings

Uma string é uma sequência de caracteres simples. Na linguagem Python, as strings são utilizadas com aspas simples ('... ') ou aspas duplas ("..."). Para exibir uma string, utiliza-se o comando print().

```
print("Olá Mundo!")

print('Olá Mundo!')
```

Concatenação de strings

Para concatenar strings, utiliza-se o operador +.

```
print("Tutorial"+"Python")  # TutorialPython
print("Tutorial "+"Python") # Tutorial Python
print("Tutorial"+" Python") # Tutorial Python
a = 'Programação'
b = ' em '
c = 'Python'
d = a+b+d
print(c)                    # Programação em Python
```

Manipulação de strings

Em Python, existem várias funções (métodos) para manipular strings. Na tabela a seguir são apresentados os principais métodos para a manipulação as strings.

```
frase = "programação em Python"

# Retorna o tamanho da string
print(len(frase))               # 21

# Retorna a string com a primeira letra maiúscula
print(frase.capitalize())       # Programação em python

# Informa quantas vezes um caractere (ou uma sequência de caracteres)
aparece na string
print(frase.count("m"))         # 2

# Verifica se uma string inicia com uma determinada sequência
print(frase.startswith("pro"))  # True

# Verifica se uma string termina com uma determinada sequência
print(frase.endswith("Py"))     # False

# Verifica se a string possui algum conteúdo alfanumérico (letra ou número)
palavra = "!@#$%"
print(palavra.isalnum())        # False

# Verifica se a string possui apenas conteúdo alfabético
palavra = "Python"
print(palavra.isalpha())        # True
```

```
# Verifica se todas as letras de uma string são minúsculas
palavra = "pytHon"
print(palavra.islower())        # False

# Verifica se todas as letras de uma string são maiúsculas
palavra = "PYTHON"
print(palavra.isupper())        # True

# Retorna uma cópia da string trocando todas as letras para minúsculo
print(frase.lower())            # programação em python

# Retorna uma cópia da string trocando todas as letras para maiúsculo
print(frase.upper())            # PROGRAMAÇÃO EM PYTHON

# Inverte o conteúdo da string (Minúsculo / Maiúsculo)
print(frase.swapcase())         # PROGRAMAÇÃO EM pYTHON

# Converte para maiúsculo todas as primeiras letras de cada palavra da
string
print(frase.title())            # Programação Em Python

# Transforma a string em uma lista, utilizando os espaços como referência
print(frase.split())            # ['programação', 'em', 'Python']

# replace(S1, S2) substitui na string o trecho S1 pelo trecho S2
print(frase.replace('em', 'com'))  # programação com Python

# Retorna o índice da primeira ocorrência de um determinado caractere na
string. Se o caractere não estiver na string retorna -1.
print(frase.find("a")) # 5

# Ajusta a string para um tamanho mínimo, acrescentando espaços à direita
se necessário
palavra = " Python"
print(palavra.ljust(15)) # " Python       "

# Ajusta a string para um tamanho mínimo, acrescentando espaços à esquerda
se necessário
print(palavra.rjust(15)) # "        Python"

# Ajusta a string para um tamanho mínimo, acrescentando espaços à esquerda
e à direita, se necessário
print(palavra.center(10)) # "  Python  "

# Remove todos os espaços em branco do lado esquerdo da string
palavra = "   Python   "
print(palavra.rstrip())  # "   Python"

# Remove todos os espaços em branco do lado direito da string
print(palavra.lstrip())  # "Python   "

# Remove todos os espaços em branco da string
```

```
print(palavra.strip())  # "Python"
```

Substrings (fração de uma string)

O fatiamento é um recurso usado para obter apenas uma parte dos elementos de uma string.

`nome_da_string[Limite_Inferior : Limite_Superior]`

O resultado é uma substring com os elementos das posições entre o `limite inferior` até o `limite superior - 1`.

```
palavra = "Python"

palavra[1:4]   # Seleciona os elementos das posições 1,2,3, ou seja, 'yth'

palavra[2:]    # Seleciona os elementos a partir da posição 2, ou seja,
'thon'

palavra[:4]    # Seleciona os elementos a partir do início até a posição 3,
ou seja, 'Pyth'
```

Revisão

1 – Considere a string A = "Para o infinito e além". Qual comando retorna o trecho "o infinito"?

2 - Escreva um programa que solicite uma frase ao usuário e retorne a frase toda em letras maiúsculas e sem espaços em branco.

Números

Os quatro tipos numéricos simples, utilizados em Python, são números inteiros (int), números longos (long), números decimais (float) e números complexos (complex).

A linguagem Python também possui operadores aritméticos, lógicos, de comparação e de bit.

Operadores numéricos

Tabela de Operadores Aritméticos

Operador	Descrição	Exemplo
+	Soma	5 + 5 = 10
-	Subtração	7 – 2 = 5
*	Multiplicação	2 * 2 = 4
/	Divisão	4 / 2 = 2

Operador	Descrição	Exemplo
%	Resto da divisão	10 % 3 = 1
**	Potência	4 ** 2 = 16

Tabela de Operadores de Comparação

Operador	Descrição	Exemplo
<	Menor que	a < 10
<=	Menor ou igual	b <= 5
>	Maior que	c >2
>=	Maior ou igual	d >= 8
==	Igual	e == 5
!=	Diferente	f != 12

Tabela de Operadores Lógicos

Operador	Descrição	Exemplo
Not	NÃO	not a
And	E	(a <=10) and (c = 5)
Or	OU	(a <=10) or (c = 5)

Esses operadores são usados para realizar operações lógicas em expressões booleanas.

`Operador NOT`

A	not A
True	False
False	True

`Operador AND`

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

`Operador OR`

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

Revisão

Escreva um programa que:

1. Receba o salário de um funcionário (float)
2. Receba o índice de reajuste (int). Exemplo: 5%
3. Calcule e mostre o valor que será pago como novo salário.

Listas

Lista é um conjunto sequencial de valores, onde cada valor é identificado através de um índice. O primeiro valor tem índice 0. Uma lista em Python é declarada da seguinte forma: `Nome_Lista = [ valor1, valor2, ..., valorN]`. Uma lista pode ter valores de qualquer tipo, incluindo outras listas.

```
lista = [5, 'morango', 3.1416, [1, 2, 3], "Python" , (2 , 'g')]

print(lista[2])    # 3.1416
print(lista[3][1]) # 2
print(lista[4])    # Python
```

Para alterar um elemento da lista, basta fazer uma atribuição de valor através do índice. O valor existente será substituído pelo novo valor.

```
lista[3] = 'banana'
lista[4] = 'abacate'
print(lista) # [5, 'morango', 3.1416, 'banana', 'abacate' , (2 , 'g')]

lista[10] = 'tijolo' # Observe que o índice 10 não existe e retornará o
seguinte erro
                     # IndexError: list assignment index out of range
```

Funções para manipulação de listas

A lista é uma estrutura mutável, ou seja, ela pode ser modificada. Na tabela a seguir estão algumas funções utilizadas para manipular listas.

Considere a `lista = [10, 40, 30, 20]`

Função	Descrição	Exemplo
len	Retorna o tamanho da lista	len(lista) # 4
min	Retorna o menor valor da lista	min(lista) # 10
max	Retorna o maior valor da lista	max(lista) # 40
sum	Retorna soma dos elementos da lista	sum(lista) # 100
append	Adiciona um novo valor na no final da lista	lista.append(100) # [10, 40, 30, 20, 100]
extend	Insere uma lista no final de outra lista	lista.extend([3, 4, 5]) # [10, 40, 30, 20, 100, 3, 4, 5]
del	Remove um elemento da lista, dado seu índice	del lista[1] # [10, 30, 20, 100, 3, 4, 5]
in	Verifica se um valor pertence à lista	3 in lista # True
sort	Ordena em ordem crescente	lista.sort() # [3, 4, 5, 10, 20, 30, 100]
reverse	Inverte os elementos de uma lista	lista.reverse() # [100, 30, 20, 10, 5, 4, 3]

Operações com lista

Concatenação (+)

```
a = [0,1,2]
b = [3,4,5]
c = a + b
print(c)   # [0, 1, 2, 3, 4, 5]
```

Repetição (*)

```
lista = [1,2]
resultado = lista * 4
print(resultado) # [1, 2, 1, 2, 1, 2, 1, 2]
```

Fatiamento de listas

O fatiamento de listas é semelhante ao fatiamento de strings.

```
frutas = ['uva', 'manga', 'banana', 'morango', 'abacate', 'abacaxi']

frutas[1:4] # Seleciona os elementos das posições 1,2,3, ou seja ['manga',
'banana', 'morango']

frutas[4:] # Seleciona os elementos a partir da posição 4, ou seja,
['abacate', 'abacaxi']
```

```
frutas[:3] # Seleciona os elementos até a posição 2, ou seja, ['uva',
'manga', 'banana']
```

Criação de listas com range()

A função range() define um intervalo de valores inteiros. Associada a list(), cria uma lista com os valores do intervalo.

A função range() pode ter de um a três parâmetros:

1. range(n) -> gera um intervalo de 0 a n-1
2. range(i , n) -> gera um intervalo de i a n-1
3. range(i , n, p) -> gera um intervalo de i a n-1 com intervalo p entre os números

Exemplo:

```
L = list(range(5))
print(L)   # [0, 1, 2, 3, 4]
L = list(range(3,8))
print(L)   # [3, 4, 5, 6, 7]
L = list(range(2,11,2))
print(L)   # [2, 4, 6, 8, 10]
```

Revisão

1. Dada a lista L = [5, 7, 2, 9, 4, 1, 3], escreva um programa que imprima as seguintes informações: a) tamanho da lista. b) maior valor da lista. c) menor valor da lista. d) soma de todos os elementos da lista. e) lista em ordem crescente. f) lista em ordem decrescente.

2. Gere uma lista contendo os múltipos de 7 entre 1 e 100.

Tuplas

Tupla, assim como a Lista, é um conjunto sequencial de valores, onde cada valor é identificado através de um índice. A principal diferença entre elas é que as tuplas são imutáveis, ou seja, seus elementos não podem ser alterados.

Dentre as utilidades das tuplas, destacam-se as operações de empacotamento e desempacotamento de valores.

Uma tupla em Python é declarada da seguinte forma: `nome_da_tupla = (valor1, valor2, ..., valorN)`

Exemplo:

```
T = (1,2,3,4,5)
print(T)    # (1, 2, 3, 4, 5)
```

```
print(T[3]) # 4
T[3] = 8    # Vai indicar um erro
            # TypeError: 'tuple' object does not support item assigment
```

Uma ferramenta muito utilizada em tuplas é o desempacotamento, que permite atribuir os elementos armazenados em uma tupla a diversas variáveis.

Exemplo:

```
T = (10,20,30,40,50)
a,b,c,d,e = T
print("a=",a,"b=",b) # a= 10 b= 20
print("d+e=",d+e)    # d+e= 90
```

Dicionários

Dicionário é um conjunto de valores, onde cada valor é associado a uma chave de acesso.

Um dicionário em Python é declarado da seguinte forma:

```
Nome_dicionario = { chave1 : valor1,
                    chave2 : valor2,
                    chave3 : valor3,
                    ...
                    chaveN : valorN
                  }
```

Exemplo:

```
produtos={"arroz": 17.30, "feijão":12.50,"carne":23.90,"alface":3.40}

print(produtos)           # {'arroz': 17.3, 'carne': 23.9, 'alface': 3.4,
'feijão': 12.5}

print(produtos["carne"])  # 23.9

print(produtos["tomate"]) # Vai mostrar o seguinte erro, por não existir a
chave
                          # KeyError: 'tomate'
```

É possível acrescentar ou modificar valores no dicionário:

```
produtos["carne"]=25.0

produtos["tomate"]=8.80
```

```
print(produtos) # {'arroz': 17.3, 'carne': 25.0, 'alface': 3.4, 'feijão':
12.5, 'tomate': 8.80}
```

Operações em dicionários

Os dicionários podem ter valores de diferentes tipos.

Exemplo:

```
Dx ={2:"carro", 3:[4,5,6], 7:('a','b'), 4: 173.8}

print(Dx[7])  # ('a', 'b')
```

Comando	Descrição
del	Exclui um item informando a chave
in	Verificar se uma chave existe no dicionário
keys()	Obtém as chaves de um dicionário
values()	Obtém os valores de um dicionário

```
D = {'alface':3.4, 'arroz':17.3, 'carne':25.0, 'feijão':7.90, 'tomate':8.8}

del D["feijão"]
print(D)        #  {'alface':3.4, 'arroz':17.3, 'carne':25.0,
'tomate':8.8}

"batata" in D    # False

"alface" in D    # True

D.keys()         # dict_keys(['alface', 'arroz', 'carne', 'tomate'])

D.values()       # dict_values([3.4, 17.3, 25.0, 8.8])
```

Revisão

1. Crie um dicionário que represente os produtos de uma lanchonete, onde o nome do produto é a chave e seu respectivo custo é o valor. (utilize ao menos 5 produtos)
2. Considere um dicionário com dez alunos e suas respectivas médias. Escreva um programa que calcule a média dessas notas.

Bibliotecas

As bibliotecas armazenam funções pré-definidas, que podem ser utilizados em qualquer momento do programa. Em Python, muitas bibliotecas são instaladas por padrão junto com o programa. Para usar uma biblioteca, deve-se utilizar o comando import:

```
import math
print(math.factorial(6))
```

Podemos também importar apenas uma função específica da biblioteca:

```
from math import factorial
print(factorial(6))
```

A tabela a seguir ilustra algumas bibliotecas comumente utilizadas em Python

Biblioteca	**Função**
math	Funções matemáticas
tkinter	Interface gráfica padrão
smtplib	e-mail (SMTP)
time	Funções de tempo

Além das bibliotecas padrão, existem também outras bibliotecas externas de alto nível disponíveis para Python, tais como:

Biblioteca	**Função**
urllib	Leitor de RSS para uso na Internet
numpy	Funções matemáticas mais avançadas
PIL/Pillow	Manipulação de imagens

Estruturas de decisão

As estruturas de decisão permitem alterar o curso do fluxo de execução de um programa, de acordo com o valor (Verdadeiro/Falso) de um teste lógico.

Podemos encadear condições de verificação:

```
if (se)
if..else (se..senão)
if..elif..else (se..senão..senão se)
```

Estrutura IF

O comando IF é utilizado quando precisamos decidir se um trecho do programa deve ou não ser executado. Ele é associado a uma condição, e o trecho de código será executado se o valor da condição for verdadeiro.

Sintaxe:

```
if <condição>:
  <Bloco de comandos>
```

Exemplo:

```
idade = int(input("Qua a sua idade: "))
if idade < 18:
  print('Você ainda não pode dirigir!')
```

Estrutura IF..ELSE

Nesta estrutura, um trecho de código será executado se a condição for verdadeira e outro se a condição for falsa.

Sintaxe:

```
if <condição>:
  <Bloco de comandos para condição verdadeira>
else:
  <Bloco de comandos para condição falsa>
```

Exemplo:

```
idade = int(input("Qua a sua idade: "))
if idade > 18:
  print('Você pode dirigir se tiver habilitação')
else:
  print('Você ainda não pode dirigir!')
```

Estrutura IF..ELIF..ELSE

Se houver diversas condições, cada uma associada a um trecho de código, utiliza-se o elif.

Sintaxe:

```
if <condição1>:
  <Bloco de comandos para condição 1>
elif <condição2>:
  <Bloco de comandos para condição 2>
```

```
elif <condiçãoN>:
  <Bloco de comandos para condição N>
else:
  <Bloco de comandos padrão>
```

Somente o bloco de comandos associado à primeira condição verdadeira encontrada será executado. Se nenhuma das condições tiver valor verdadeiro, executa o bloco de comandos padrão.

Revisão

1. Faça um programa que leia 2 notas de um aluno, calcule a média e imprima aprovado ou reprovado (para ser aprovado a média deve ser no mínimo 6)
2. Refaça o exercício 1, identificando o conceito aprovado (média superior a 6), exame (média entre 4 e 6) ou reprovado (média inferior a 4).

Estruturas de repetição

A Estrutura de repetição é utilizada para executar uma mesma sequência de comandos várias vezes. A repetição está associada ou a uma condição, que indica se deve continuar ou não a repetição, ou a uma sequência de valores, que determina quantas vezes a sequência deve ser repetida. As estruturas de repetição são conhecidas também como laços (loops).

Laço WHILE

No laço while, o trecho de código da repetição está associado a uma condição. Enquanto a condição tiver valor verdadeiro, o trecho é executado. Quando a condição passa a ter valor falso, a repetição termina.

Sintaxe:

```
while <condição_verdadeira>:
  <Bloco de comandos>
```

Exemplo:

```
senha = '12345'
leitura = ''
while (leitura != senha):
  leitura = input("Digite a senha: ")
  if leitura == senha:
    print('Acesso liberado!')
    break
  else:
    print('Senha incorreta! Tente novamente.')
```

Laço FOR

O laço for é a estrutura de repetição mais utilizada em Python. Pode ser utilizado com uma sequência numérica (gerada com o comando range) ou associado a uma lista. O trecho de código da repetição é executado para cada valor da sequência numérica ou da lista.

Sintaxe:

```
for <variável> in range (início, limite, passo):
  <Bloco de comandos >
```

ou

```
for <variável> in <lista> :
  <Bloco de comandos >
```

Exemplo:

```
# Encontre a soma dos dez primeiros números da sequência: 1, 4, 7, 10, 13,
16, 19, 21, 24, 27
soma=0
for x in range(1,30,3):
  some = soma + x
print('Soma = ', soma)
```

Revisão

1. Escreva um programa que receba as notas dos alunos e armazene em uma lista. Ao receber '-1', finalize a leitura de notas e calcule a média da sala.
2. Escreva um programa para encontrar a soma dos valores: S = 3 + 6 + 9 + ... 333
3. Escreva um programa que receba números inteiros de 1 a 10 e mostre a tabuada desse número.

Funções

Funções são definidas usando a palavra-chave DEF

Sintaxe:

```
def nome_da_função(parâmetros):
  <Bloco de comandos>
```

Exemplo:

```
def hello():
  print('Olá Mundo!')

hello()    # Olá Mundo!
```

Parâmetros e argumentos

Parâmetros são as variáveis que podem ser incluídas nos parênteses das funções. Quando a função é chamada são passados valores para essas variáveis. Esses valores são chamados argumentos. O corpo da função pode utilizar essas variáveis, cujos valores podem modificar o comportamento da função.

Exemplo:

```
def soma(a,b):
  print(a+b)

soma(3,5)    # 8
```

Escopo das variáveis

Toda variável utilizada dentro de uma função tem escopo local, isto é, ela não será acessível por outras funções ou pelo programa principal. Se houver variável com o mesmo nome fora da função, será uma outra variável, completamente independentes entre si.

Exemplo:

```
def soma(x,y):
  total = x+y
  print("Total soma = ",total)  # 8

# programa principal
total = 10
soma(3,5)
print("Total principal = ",total) # 10
```

Para uma variável ser compartilhada entre diversas funções e o programa principal, ela deve ser definida como variável global. Para isto, utiliza-se a instrução global para declarar a variável em todas as funções para as quais ela deva estar acessível. O mesmo vale para o programa principal.

Exemplo:

```
def soma(x,y):
  global total
  total = x+y
  print("Total soma = ",total) # 8
```

```
#programa principal
global total
total = 10
soma(3,5)
print("Total principal = ",total) # 8
```

Retorno de valores

O comando return é usado para retornar um valor de uma função e encerrá-la. Caso não seja declarado um valor de retorno, a função retorna o valor None (que significa nada, sem valor).

Exemplo:

```
def soma(x,y):
  total = x+y
  return total

#programa principal
s=soma(3,5)
print("soma = ",s) # 8
```

Observações:

a) O valor da variável total, calculado na função soma, retornou da função e foi atribuído à variável s.

b) O comando após o return foi ignorado.

Valor padrão

É possível definir um valor padrão para os parâmetros da função. Neste caso, quando o valor é omitido na chamada da função, a variável assume o valor padrão.

Exemplo:

```
def calcula_juros(valor, taxa=10):
  juros = valor*taxa/100
  return juros

calcula_juros(500)   # 50.0
```

Revisão

1. Crie uma função para desenhar uma linha, usando o caractere '_'. O tamanho da linha deve ser definido na chamada da função.
2. Crie uma função que receba como parâmetro uma lista, com valores de qualquer tipo. A função deve imprimir todos os elementos da lista numerando-os.

3. Crie uma função que receba como parâmetro uma lista com valores numéricos e retorne a média desses valores.

Referências

[1] Van Rossum, Guido. Repositório no GitHub. Disponível em: https://github.com/gvanrossum

[2] Python Software Foundation. Disponível em: https://www.python.org/psf-landing/

[3] O tutorial de Python. Disponível em: https://docs.python.org/pt-br/3/tutorial/

[4] Van Rossum, Guido. Divagações sobre tecnologia, política, cultura e filosofia pelo criador da linguagem de programação Python. Disponível em: https://neopythonic.blogspot.com/2016/04/kings-day-speech.html

Outros conteúdos foram compilados de apostilas disponíveis na Internet

Agradecimentos a: Alura, Grupo PET IFSP São Carlos, Prof. Santos, A.W.S. entre outros.