1 INTRODUÇÃO

1.1 Características da linguagem Python

A linguagem de programação Python foi criada em 1991 por Guido Van Rossumem, com a finalidade de ser uma linguagem simples e de fácil compreensão. Apesar de simples, Python é uma linguagem muito poderosa, que pode ser usada para desenvolver e administrar grandes sistemas.

Uma das principais características que diferencia a linguagem Python das outras é a legibilidade dos programas escritos. Isto ocorre porque, em outras linguagens, é muito comum o uso excessivo de marcações (ponto ou ponto e vírgula), de marcadores (chaves, colchetes ou parênteses) e de palavras especiais (begin/end), o que torna mais difícil a leitura e compreensão dos programas. Já em Python, o uso desses recursos é reduzido, deixando a linguagem visualmente mais limpa, de fácil compreensão e leitura.

Entre outras características existentes na linguagem Python, destaca-se a simplicidade da linguagem, que facilita o aprendizado da programação. Python também possui uma portabilidade muito grande para diversas plataformas diferentes, além de ser possível utilizar trechos de códigos em outras linguagens.

Python 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 VARIÁVEIS

2.1 Tipos de dados básicos

Variáveis são pequenos espaços de memória, utilizados para armazenar e manipular dados. Em Python, os tipos de dados básicos são: **tipo inteiro** (armazena números inteiros), **tipo float** (armazena números em formato decimal), e **tipo string** (armazena um conjunto de caracteres). Cada variável pode armazenar apenas um tipo de dado a cada instante.

Em Python, diferentemente de outras linguagens de programação, não é preciso declarar de que tipo será cada variável no início do programa. Quando se faz uma atribuição de valor, automaticamente a variável se torna do tipo do valor armazenado, como apresentado nos exemplos a seguir:

Exemplos:

A variável **a** se torna uma variável do tipo inteiro.

```
>>> a = 10
>>> a
10
```

A variável **b** se torna uma variável do tipo float.

```
>>>
>>> b = 2.3
>>> b
2.3
```

A variável **c** se torna uma variável do tipo string.

```
>>> c = "Olá mundo"
>>> c
'Olá mundo'
```

ou

```
>>> c = 'Olá mundo'
>>> c
'Olá mundo'
```

A variável d se torna uma variável do tipo lógico.

```
>>> d = True
>>> d
True
```

ou

```
>>> d = False
>>> d
False
```

2.2 Comando de entrada de dados

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

Exemplo:

```
>>> nome = input('Entre com o seu nome: ')
...
Entre com o seu nome: Diogenes
>>>
>>> nome
...
'Diogenes'
```

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.

Exemplos:

```
>>> num = int(input('Entre com um número: '))
...
Entre com um número: 45
>>> num
...
45
```

```
>>> altura = float(input('Entre com sua altura: '))
...
Entre com sua altura: 1.75
>>> altura
...
1.75
```

2.3 Comando de Saída de dados

Para mostrar o valor de uma variável ou de um objeto em Python podemos fazer uso do comando ***print()***.

```
>>> s = 20
>>> print(s)
20
```

O comando **print()** é a única forma de visualizarmos alguma coisa no terminal quando estamos executando um programa via arquivo fonte.

2.4 Funções de conversão de base

Python oferece várias funções embutidas e métodos para trabalhar com números em diferentes bases numéricas, como hexadecimal, binária e octal.

- **hex(x)**: Converte um número inteiro em sua representação hexadecimal como uma string.
- **bin(x)**: Converte um número inteiro em sua representação binária como uma string.
- **oct(x)**: Converte um número inteiro em sua representação octal como uma string.

2.5 Identificadores

Em Python, os nomes das variáveis devem ser iniciados com uma letra, mas podem possuir outros tipos de caracteres, como números e símbolos. O símbolo sublinha (_) também é aceito no início de nomes de variáveis.

Tabela 1 – Exemplos de nomes válidos e inválidos.

Nome	Válido	Comentários
ac1	Sim	Embora contenha um número, o nome ac1 inicia com letra.
código	Sim	Nome formado com letras.
codigo90	Sim	Nome formado por letras e números, mas inicia com letras.
Salario_medio	Sim	O símbolo (_) é permitido e facilita a leitura de nomes grandes.
Salario médio	Não	Nomes de variáveis não podem conter espaços em branco.
_salario	Sim	O sublinha (_) é aceito em nomes de variáveis, mesmo no início.
8a	Não	Nomes de variáveis não podem começar com números.

2.6 Exercícios: Variáveis

1) Escreva mais 3 identificadores que não possam ser usados como nome de variáveis.

2) Experimente as funções de conversão de base.

3) Experimente também a forma de formatação de string para converter números entre diferentes bases.

3 STRINGS

Uma **string** é uma sequência de caracteres simples. Na linguagem Python, as strings são utilizadas com aspas simples ('... ') ou aspas duplas (''... '').

Para se exibir uma string utiliza se o comando **print()**.

Exemplo:

```
>>> print('Olá mundo')
Olá mundo
```

Ou

```
>>> print("Olá mundo")
Olá mundo
```

3.1 Concatenação de strings

Para concatenar strings, utiliza-se o operador +.

Exemplo:

```
>>> print("Apostila"+"Python")
ApostilaPython
```

Ou

```
>>> a = 'Programação'
>>> b = 'Python'
>>> c = a+b
>>> print(c)
ProgramaçãoPython
```

Para se concatenar a mesma string múltiplas vezes, utiliza-se o operador *.

Exemplo:

```
>>> b*5
'PythonPythonPythonPythonPython'
```

3.2 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.

Tabela 2 - Manipulação de strings

Método	Descrição	Exemplo

len()	Retorna o tamanho da string.	`teste = "Apostila de Python"`
		`len(teste)`
		`18`

capitalize()	Retorna a string com a primeira letra maiúscula	`a = "python"`
		`a.capitalize()`
		`'Python'`

count()	Informa quantas vezes um caractere (ou uma sequência de caracteres) aparece na string.	`b = "Linguagem Python"`
		`b.count("n")`
		`2`

startswith()	Verifica se uma string inicia com uma determinada sequência.	`c = "Python"`
		`c.startswith("Py")`
		`True`

endswith()	Verifica se uma string termina com uma determinada sequência.	`d = "Python"`
		`d.endswith("Py")`
		`False`

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

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

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

isupper()	Verifica se todas as letras de uma string são maiúsculas.	`h = "# PYTHON 12"`
		`h.isupper()`
		`True`

lower()	Retorna uma cópia da string trocando todas as letras para minúsculo.	`i = "#PYTHON 3"`
		`i.lower()`
		`'#python 3'`

upper()	Retorna uma cópia da string trocando todas as letras para maiúsculo.	`j = "Python"`
		`j.upper()`
		`'PYTHON'`

swapcase()	Inverte o conteúdo da string (Minúsculo / Maiúsculo).	`k = "Python"`
		`k.swapcase()`
		`'pYTHON'`

title()	Converte para maiúsculo todas as primeiras letras de cada palavra da string.	`l = "apostila de python"`
		`l.title()`
		`'Apostila De Python'`

replace(S1,S2)	Substitui na string o trecho S1 pelo trecho S2.	`n = "Apostila teste"`
		`n.replace("teste", "Python")`
		`'Apostila Python'`

find()	Retorna o índice da primeira ocorrência de um determinado caractere na string. Se o caractere não estiver na string retorna -1.	`o = "Python"`
		`o.find("h")`
		`3`

ljust()	Ajusta a string para um tamanho mínimo, acrescentando espaços à direita se necessário.	`p = "Python"`
		`p.ljust(15)`
		`'Python         '`

rjust()	Ajusta a string para um tamanho mínimo, acrescentando espaços à esquerda se necessário.	`p = "Python"`
		`p.rjust(15)`
		`'      Python '`

center()	Ajusta a string para um tamanho mínimo, acrescentando espaços à esquerda e à direita, se necessário.	`p = "Python"`
		`p.center(15)`
		`'    Python    '`

lstrip()	Remove todos os espaços em branco do lado esquerdo da string.	`s = " Python "` `s.lstrip()`

		'Python '
rstrip()	Remove todos os espaços em branco do lado direito da string.	s = " Python " s.rstrip() ' Python'
strip()	Remove todos os espaços em branco da string.	s = " Python " s.strip() 'Python'

3.3 Fatiamento de strings

O fatiamento é uma ferramenta usada para extrair apenas uma parte dos elementos de uma string;

Nome_String [Limite_Inferior : Limite_Superior]

Retorna uma string com os elementos das posições do limite inferior até o limite superior – 1.

Exemplo:

```
>>> s = 'Programação Orientada a Objetos'
```

Seleciona os elementos das posições 0 até 10:

```
>>> s[0:11]
'Programação'
```

Seleciona os elementos a partir da posição 12:

```
>>> s[12:]
'Orientada a Objetos'
```

Seleciona os elementos até a posição 5:

```
>>> s[:6]
'Progra'
```

3.4 Exercícios: Strings

1) Considere a string A = "Um elefante incomoda muita gente". Que fatia corresponde a "elefante incomoda"?

2) Escreva um programa que solicite uma frase ao usuário e reescreva a frase toda em maiúscula e sem espaços em branco.

3) Escreva um programa que solicite uma frase ao usuário e reescreva a frase usando "leet speak". Por exemplo, hello -> h3110, leet -> 1337, gaming -> g4m1ng, password -> p455w0rd. Utilize a função replace neste caso.

4 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.

4.1 Operadores numéricos

Tabela 3 - Operadores Aritméticos

Operador	Descrição	Exemplo
+	Soma	5 + 6 = 11
-	Subtração	7 – 2 = 5
*	Multiplicação	2 * 2 = 4
//	Divisão inteira	7 // 2 = 3
%	Resto da divisão inteira	10 % 3 = 1
/	Divisão real	7 / 2 = 3.5
**	Potencia	4 ** 2 = 16

Tabela 4 - Operadores Relacionais

Operador	Descrição	Exemplo
==	Igualdade	A == 5
!=	Diferença	B != 12
<	Menor que	C < 10
<=	Menor ou igual a	D <=7
>	Maior que	E > 2
>=	Maior ou igual a	F >= 8

Tabela 5 - Operadores Lógicos

Operador	Descrição	Exemplo
not	NÃO	not a
and	E	(a<=10) and (b==5)
or	OU	(a<=10) or (b==5)

4.2 Exercícios: números

1) Escreva um programa que receba 2 valores do tipo inteiro x e y, e calcule o valor de z:

$$z = \frac{(x^2 + y^2)}{(x - y)^2}$$

2) Escreva um programa que receba o salário de um funcionário (float), e retorne o resultado do novo salário com reajuste de 35%.

3) Escreva um programa que ache as raízes da equação $2x^2 – 18x + 12$.

5 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]

Exemplo:

```
>>> l1 = [1,2,3,4]
```

- Lista inteira:

```
>>> print(l1)
[1, 2, 3, 4]
```

- Somente o elemento 2 da lista:

```
>>> print(l1[2])
3
```

Uma lista pode ter valores de qualquer tipo, incluindo outras listas.

Exemplos:

```
>>> l2 = [3,'abacate',9.7,[3,4,5],"Python"]
```

- Elemento 1 da lista l2:

```
>>> print(l2[1])
abacate
```

- Elemento 3 da lista l2, que é outra lista:

```
>>> print(l2[3])
[3, 4, 5]
```

- Elemento 1 da lista que está em l2[3]:

```
>>> print(l2[3][1])
4
```

- A tentativa de acesso a um índice inexistente resultará em erro.

```
>>> print(l2[9])
Traceback (most recent call last):
  File "<pyshell#54>", line 1, in <module>
    print(l2[9])
IndexError: list index out of range
```

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

Exemplo:

```
>>> l2[3] = 'morango'
>>> print(l2)
[3, 'abacate', 9.7, 'morango', 'Python']
```

5.1 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.

Tabela 6 – Operações com listas

Função	Descrição	Exemplo
len	retorna o tamanho da lista.	L = [1, 2, 3, 4] len(L) 4
min	retorna o menor valor da lista.	L = [30, 40, 10, 20] min(L) 10
max	retorna o maior valor da lista.	L = [30, 40, 10, 20] max(L) 40
sum	retorna soma dos elementos da lista.	L = [10, 20, 30] sum(L) 60
append	adiciona um novo valor na no final da lista.	L = [1, 2, 3] L.append(100) L [1, 2, 3, 100]
extend	insere uma lista no final de outra lista.	L = [0, 1, 2] L.extend([3, 4, 5]) L [0, 1, 2, 3, 4, 5]
del	remove um elemento da lista, dado seu índice.	L = [1,2,3,4] del L[1] L [1, 3, 4]
in	verifica se um valor pertence à lista.	L = [1, 2 , 3, 4] 3 in L True
sort()	ordena em ordem crescente	L = [3, 5, 2, 4, 1, 0] L.sort() L [0, 1, 2, 3, 4, 5]
reverse()	inverte os elementos de uma lista.	L = [0, 1, 2, 3, 4, 5] L.reverse() L [5, 4, 3, 2, 1, 0]

5.2 Operadores com listas

Concatenação (+)

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

Repetição (*)

```
>>> L = [1,2]
>>> R = L*4
>>> print(R)
[1, 2, 1, 2, 1, 2, 1, 2]
```

5.3 Fatiamento de listas

O fatiamento de listas é semelhante ao fatiamento de strings.

Exemplo:

```
>>> L = [0,1,2,3,4,5,6]

>>> L[1:4]
[1, 2, 3]

>>> L[2:]
[2, 3, 4, 5, 6]

>>> L[:4]
[0, 1, 2, 3]
```

5.4 Listas e strings

Podemos separar as partes de uma string e criar uma lista.

Função	Descrição	Exemplo

split()	Transforma a string em uma lista, utilizando os espaços como referência.	`m = "cana de açúcar"`
		`m.split()`
		`['cana', 'de', 'açúcar']`

5.5 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 1 a 3 parâmetros:

- **range(n)** : gera um intervalo de 0 a n-1
- **range(i , n)** : gera um intervalo de i a n-1
- **range(i , n, p)** : gera um intervalo de i a n-1 com intervalo p entre os números

Exemplos:

```
>>> L1 = list(range(5))
>>> L1
[0, 1, 2, 3, 4]

>>> L2 = list(range(3,8))
>>> L2
[3, 4, 5, 6, 7]

>>> L3 = list(range(2,11,3))
>>> L3
[2, 5, 8]
```

5.6 Exercícios: listas

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 de contendo os múltiplos de 3 entre 1 e 50.

6 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.

Em Python temos as seguintes estruturas de decisão

- **if** (se)
- **if...else** (se...senão)
- **if...elif...else** (se...senão...senão se)

6.1 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:

```
valor = int(input("Qual sua idade?"))
if valor < 18:
    print("Você ainda não pode dirigir!")
```

6.2 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:

```
valor = int(input("Qual sua idade?"))
if valor < 18:
    print("Você ainda não pode dirigir!")
else:
    print("Você é o cara!")
```

6.3 Comando if ..e lif ..else

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

Sintaxe:

```
if <condição1> :
        <Bloco de comandos 1>
elif <condição2> :
        <Bloco de comandos 2>
elif <condição3> :
        <Bloco de comandos 3>
        ::::::::::::::::::::::::::::::::::::::
else :
        <Bloco de comandos default>
```

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 default.

Exemplo:

```
valor = int(input("Qual sua idade?"))
if valor < 6:
    print("Que coisa fofa!");
elif valor < 18:
    print("Você ainda não pode dirigir!")
elif valor > 70:
    print("Prefira andar de Uber!")
else:
    print("Você é o cara!")
```

6.4 Exercícios: estruturas de decisão

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).

3) Dados 3 valores A, B, C verificar se eles podem ser os comprimentos dos lados de um triângulo e, se forem, verificar se o triângulo é equilátero, isósceles ou escaleno.

   Para um polígono ser um triângulo, o comprimento de cada um de seus três lados deve ser menor do que a soma dos outros dois.

A seguir, classifique-o:

- Equilátero – 3 lados iguais
- Isósceles – 2 lados iguais
- Escaleno – 3 lados diferentes

4) Escreva um programa que recebe o peso e a altura de uma pessoa, e calcula seu IMC pela formula peso / (altura)^2 , e mostre o resultado baseado nas seguintes informações:
   - IMC abaixo de 20 – peso leve
   - entre 20 e 25 – peso normal
   - entre 25 e 30 – excesso de peso
   - entre 30 e 35 – obesidade leve
   - entre 35 e 40 – obesidade moderada
   - acima de 40 – obesidade mórbida

7 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).

Em Python temos as seguintes estruturas de repetição

- **while** (enquanto-faça)
- **for** (para-faça)

7.1 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> :
<Bloco de comandos>

Exemplo:

```
senha = "54321"
leitura = " "
while (leitura != senha):
    leitura = input ("Digite a senha: ")
    if leitura == senha :
        print('Acesso liberado!')
    else:
        print('Senha incorreta. Tente novamente!')
```

```
Digite a senha: abcd
Senha incorreta. Tente novamente!
Digite a senha: 12345
Senha incorreta. Tente novamente!
Digite a senha: 54321
Acesso liberado!
```

Exemplo: Calcular a soma de 5 valores.

```
cont = 0
soma = 0
while cont < 5:
    cont = cont + 1
    valor = float(input('Digite o '+str(cont)+'-ésimo valor: '))
    soma = soma + valor

print('Soma = ', soma)
```

```
Digite o 1-ésimo valor: 5
Digite o 2-ésimo valor: 6
Digite o 3-ésimo valor: 7
Digite o 4-ésimo valor: 8
Digite o 5-ésimo valor: 9
Soma =  35.0
```

7.2 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>

Exemplos:

Encontrar a soma S = 1+4+7+10+13+16+19

```
soma = 0
for x in range(1,20,3):
    soma = soma + x
print('Soma = ', soma)
```

As notas de um aluno estão armazenadas em uma lista. Calcular a média dessas notas.

```
Lista_notas= [3.4,6.6,8,9,10,9.5,8.8,4.3]
soma=0
for nota in Lista_notas:
    soma = soma + nota

média = soma / len(Lista_notas)
print('Média = ', média)
```

7.3 Exercícios: estrutura de repetição

1) Escreva um programa para encontrar a soma S = 3 + 6 + 9 + …. + 333.

2) Escreva um programa que leia 10 notas e informe a média dos alunos.

3) Escreva um programa que leia um número de 1 a 10, e mostre a tabuada desse número.

4) Escreva um programa em Python que conte o número de vogais em uma string fornecida pelo usuário. O programa deve solicitar ao usuário que insira uma string e, em seguida, conte quantas vogais (as letras 'a', 'e', 'i', 'o', 'u' em minúsculas ou maiúsculas) estão presentes na string. Conte separadamente e também apresente o valor total (a soma de todas elas).

8 FUNÇÕES

Funções são pequenos trechos de código reutilizáveis. Elas permitem dar um nome a um bloco de comandos e executar esse bloco, a partir de qualquer lugar do programa.

8.1 Como definir uma função

Funções são definidas usando a palavra-chave **def**, conforme sintaxe a seguir:

def <nome_função> (<definição dos parâmetros >) :
<Bloco de comandos da função>

Obs.: A definição dos parâmetros é opcional.

Exemplo: Função simples

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

Para usar a função, basta chamá-la pelo nome:

```
>>> hello()
Olá Mundo!!!
```

8.2 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: Função para imprimir o maior entre 2 valores

```
def maior(x,y):
    if x > y:
        print(x)
    else:
        print(y)
```

```
>>> maior(4,7)
7
```

8.3 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)

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

Resultado da execução:

```
Total soma =  8
Total principal =  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)

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

Resultado da execução:

```
Total soma =  8
Total principal =  8
```

8.4 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)
```

Resultado da execução:

```
soma =  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.

8.5 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
```

Resultado da execução:

```
>>> calcula_juros(8)
0.8
```

8.6 Exercícios: funções

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.

9 BIBLIOGRAFIA

BEAZLEY, D. ; JONES, B.K. **Python Cookbook.** Ed. Novatec, 2013.

BORGES, L. E. **Python para desenvolvedores.** 1ed. São Paulo SP: Novatec, 2014.

GRUPO PET TELE. **Tutorial de introdução ao Python.** Niterói RJ: Universidade Federal Fluminense (UFF) / Escola de Engenharia, 2011. Apostila

LABAKI, J. **Introdução a python Módulo A.** Ilha Solteira SP: Universidade Estadual Paulista (UNESP), 2011. Apostila

MENEZES, N. N. C. **Introdução à programação com python.** 2ed. São Paulo SP:Novatec, 2014.

PYTHON. **Python Software Foundation.** Disponível em: <https://www.python.org/>. Acesso em: dezembro de 2023.