**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).

1.2 Instalação do interpretador Python

a) Instalação de Python no Linux Nas versões mais recentes do GNU/Linux, o Python já se encontra instalado, bastando ao programador entrar no terminal e digitar python. Caso não esteja, seguem os passos para a instalação no terminal:

1. Acesse o terminal Linux. 3 2. Digite o comando sudoapt-getinstall python3.4 no terminal do GNU/Linux para inicializar o processo de instalação. 3. Terminado o download, o interpretador já estará instalado no computador.

b) Instalação do IDLE no Linux O IDLE é um ambiente integrado de desenvolvimento que acompanha a instalação do interpretador Python em sistemas operacionais Windows. Para tê-lo disponível em distribuições Linux basta seguir as etapas abaixo:

1. Acesse o terminal Linux.

2. Digite o comando sudoapt-getinstall idle-python3.4.

3. Para executá-lo basta digitar no terminal idle-python3.4 &.

4

c) Instalação do Python no Windows

A instalação do interpretador Python para Windows é mais simples, conforme apresentado a seguir:

1. Entre no site www.python.org. Na aba download selecione a versão atual.

2. Após o download, executa o instalador mantendo, por default, todas as configurações a cada passo da instalação. Depois clique em Finalizar e o interpretador Python já estará instalado no computador. Caso você não consiga executar o interpretador Python pelo prompt de comando, provavelmente o path não está configurado. Veja abaixo os passos para configurá-lo:

1. Com o cursor do mouse vá até Computador, clique com o botão direito e escolha Propriedades.

2. Depois clique em Configurações avançadas do sistema e, a seguir, Variáveis de ambiente.

3. Com ajuda da barra de rolagem procure a variável chamada path, selecione-a e escolha a opção Editar.

4. Na próxima janela, no campo Valor de variável, você irá encontrar uma lista contendo vários paths de outros programas. Para adicionar um novo path, vá até o final da lista e acrescente um ponto e vírgula ( ; ). Depois disso, copie o endereço da pasta onde se encontra instalado o interpretador Python e cole após ponto e vírgula.

**2. VARIÁVEIS**

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=10

>>>a

10

A variável a se torna uma variável do tipo inteiro.

>>>b=1.2

>>>b

1.2

A variável b se torna uma variável do tipo float.

>>>c=”Olá Mundo”

>>>c

‘Olá Mundo´

A variável c se torna uma variável do tipo string.

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:



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:





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** |
| a3 | Sim | Embora contenha um número, o nome a3 inicia com letra. |
| velocidade | Sim | Nome formado com letras. |
| velocidade90 | Sim | Nome formado por letras e números, mas inicia com letras. |
| salario\_médio | Sim | O símbolo ( ) é permitido e facilita a leitura de nomes grandes. |
| salario médio | Não | Nomes de variáveis não podem conter espaços em branco. |
| \_salário | Sim | O sublinha ( ) é aceito em nomes de variáveis, mesmo no início. |
| 5A | Não | Nomes de variáveis não podem começar com números. |

**3. 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().

Exemplo:



 3.1**Concatenação de strings** Para concatenar strings, utiliza-se o operador +. Exemplo:



**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' |
| split() | Transforma a string em uma lista, utilizando os espaços como referência. | >>> m = "cana de açúcar"  >>>m.split()  ['cana', 'de', 'açúcar'] |
| 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. | >>>q = "Python"  >>>q.rjust(15)  ' Python' |
| center() | Ajusta a string para um tamanho mínimo, acrescentando espaços à esquerda e à direita, se necessário. | >>>r = "Python"  >>>r.center(10)  ' 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. | >>>t = " Python "  >>>t.rstrip()  ' Python' |
| strip() | Remove todos os espaços em branco da string. | >>>u = " Python "  >>>u.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 = "Python"

>>>s[1:4] seleciona os elementos das posições 1,2,3

'yth'

>>>s[2:]seleciona os elementos a partir da posição 2

'thon'

>>>s[:4] seleciona os elementos até a posição 3

'Pyth'

**3.4 Exercícios strings:**

1**–**Considere a string A = "Um elefante incomodamuita gente". Que fatia corresponde a "elefante incomoda"?

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

**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.

**4.1Operadores numéricos**

Tabela 3 - Operadores Aritméticos

|  |  |  |
| --- | --- | --- |
| **Operador** | **Descrição** | **Exemplo** |
| + | Soma | 5 + 5 = 10 |
| - | Subtração | 7 – 2 = 5 |
| \* | Multiplicação | 2 \* 2 = 4 |
| / | Divisão | 6 / 2 = 3 |
| % | Resto da Divisão | 5 % 2 = 1 |
| \*\* | Potência | 4 \*\* 2 = 16 |
| = | Atribuição | X=20 |

Tabela 4– Operadores Relacionais

|  |  |  |
| --- | --- | --- |
| **Operador** | **Descrição** | **Exemplo** |
| < | Menor | A < 5 |
| <= | Menor igual | B <= 20 |
| > | Multiplicação | C >30 |
| >= | Divisão | D >= 40 |
| != | Resto da Divisão | E != 26 |
| == | Potência | F == 13 |

Tabela 5 - Operadores Lógicos

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

**5. 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.

EmPython temos as seguintes estruturas de decisão:

**if**(se)

**if..else**(se..senão)

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

**5.1 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:**



**5.2 Estrutura if..else**

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

**Sintaxe:**

if<condição>**:**

<Blocode comandos paracondição verdadeira>

else**:**

<Blocode comandos para condição falsa>

**Exemplo:**



**5.3Comandoif..elif..else**

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

**Sintaxe:**

if<condição1>**:**

<Blocode comandos 1>

elif<condição2>**:**

<Blocode comandos 2>

elif<condição3>**:**

<Bloco de comandos 3>

:::::::::::::::::::::::::::::::::::::::::

else**:**

<Blocode comandos default>

Somenteoblocodecomandosassociadoàprimeiracondiçãoverdadeiraencontradaseráexecutado.Senenhumadascondiçõestivervalorverdadeiro,executaoblocodecomandos*default*.

**Exemplo:**

>>> valor=int(input("Qual sua idade??"))

Qual sua idade??

>>>if valor < 6:

print("aaaa")

elif valor < 18:

print("bbbbb")

elif valor > 60:

print("melhor idade")

else:

print("vc é o cara")

**6. 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***).

**6.1 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="123"

>>> leitura=""

>>>while(leitura != senha):

leitura=input("digite a senha")

if leitura==senha:

print('Acesso Liberado')

else:

print('Senha incorreta, tente novamente...')

**6.2 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:**

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

S=0

**For** x **in range**(1,20,3):

S = S+x

**print**('Soma = ',S)

2. 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

media = soma/**len**(Lista\_notas)

**print**(“Média= “,media)

**7. BIBLIOTECAS**

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

**Exemplo:importar a biblioteca de funções matemáticas:**

>>>importmath

>>>print(math.factorial(5))

120

Pode-se importar uma função específicada biblioteca:

>>>from math import factorial

>>> print (factorial(5))

120

A tabela a seguir, mostra algumas das bibliotecas padrão dePython.

|  |  |
| --- | --- |
| **Bibliotecas** | **Função** |
| math | Funções matemáticas |
| tkinter | Interface Gráfica padrão |
| smtplib | e-mail |
| time | Funções de tempo |

Algumas bibliotecas externas para Python

|  |  |
| --- | --- |
| **Bibliotecas** | **Função** |
| urllib | Leitor de RSS para uso na internet |
| numpy | Funções matemáticas mais avançadas |
| PIL/Pillow | Manipulação de imagens |

**7. LISTAS**

Lista é um conjunto sequêncial de valores, onde cada valor é identificado através de um índice. O primeiro valor tem índice0. Uma lista em Python declarada da seguinte forma:

**Nome\_Lista= [ valor1, valor2, ..., valorN]**

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

**Exemplo:**

L = [3,'abacate',9.7,[5,6,3],"Python",(3,'j')]

print(L[2])

**9.7**

print(L[3])

**[5,6,3]**

print(L[3][1])

**6**

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

**Exemplo:**

L[3]= 'morango'

print(L)

**L = [3 , 'abacate' , 9.7 , 'morango', "Python" , (3 , 'j')]**

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

L[7]= 'banana'

**7.1Funções 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.

|  |  |  |
| --- | --- | --- |
| **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 = [10, 40, 30, 20]  min(L)  10 |
| **max** | retorna o maior valor da lista. | L = [10, 40, 30, 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] |

**7.2Operações comlistas**

**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]

**7.3Fatiamento de listas**

O fatiamento de listas é semelhante ao fatiamento de strings.

Exemplo:

L = [3,'abacate',9.7,[5,6,3],"Python",(3,'j')]

L[1:4] seleciona os elementos das posições 1,2,3

['abacate', 9.7, [5, 6, 3]]

L[2:]seleciona os elementos a partir da posição 2

[9.7, [5, 6, 3], 'Python', (3, 'j')]

L[:4] seleciona os elementos até a posição 3

[3, 'abacate', 9.7, [5, 6, 3]]

**7.4Criação de listas com range()**

A função range()define um intervalo de valores inteiros.Associada a list(), criauma lista com os valores dointervalo.

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))

print(L1)

[0, 1, 2, 3, 4]

L2 = list(range(3,8))

print(L2)

[3, 4, 5, 6, 7]

L3 = list(range(2,11,3))

print(L3)

[2, 5, 8]

**7.5Exercícios:listas**

**1 –**Dada a lista L= [5, 7, 2, 9, 4,1, 3],escreva um programa que imprima as seguintes informações:

a) tamanho dalista.

b) maior valor da lista.

c) menor valor da lista.

d) soma de todos os elementos dalista.

e)lista em ordem crescente.

f) lista em ordem decrescente.

**2 –**Gere uma lista de contendo os múltiplos de 3 entre 1 e50.

**8. TUPLAS**

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

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

Uma tuplaem Pythoné declaradada seguinte forma:

**Nome\_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

Traceback (mostrecentcalllast):

File "C:/Python34/teste.py", line 4, in <module>

T[3] = 8

TypeError: 'tuple' object does notsupport item assignment

Umaferramentamuitoutilizada em tuplas é o **desempacotamento**, quepermite 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

**9. DICIONÁRIOS**

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

Um dicionário em Pythoné declaradoda seguinte forma:

Nome\_dicionario = { chave1: valor1,

chave2: valor2,

chave3: valor3,

......

chaveN: valorN}

**Exemplo:**

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

print(D)

{'arroz': 17.3, 'carne': 23.9, 'alface': 3.4, 'feijão': 12.5}

print(D["carne"])

23.9

print(D["tomate"])

Traceback (mostrecentcalllast):

File "C:/Python34/teste.py", line 4, in <module>

print(D["tomate"])

KeyError: 'tomate'

É possível acrescentar ou modificar valores no dicionário:

D["carne"]=25.0

D["tomate"]=8.80

print(D)

{'alface':3.4,'tomate':8.8,'arroz':17.3,'carne':25.0,'feijão':12.5}

Os valores do dicionário não possuem ordem, por isso a ordem de impressão dos valores não é sempre a mesma.

**9.1 Operações em dicionários**

Na tabela são apresentados alguns comandos para a manipulação de dicionários.

|  |  |  |  |
| --- | --- | --- | --- |
| **Comando** | **Descrição** | **Exemplo** | |
| del | Exclui um item informando a chave. | del D["feijão"]  print(D) {'alface':3.4 tomate':8.8,'arroz':17.3,'carne':25.0} | |
| in | Verificar se uma chave existe no dicionário. | “batata" in D  False | "alface" in D  True |
| keys() | Obtém as chaves de um dicionário. | D.keys()  dict\_keys(['alface', 'tomate,'carne', 'arroz']) | |
| values() | Obtém os valores de um dicionário. | D.values()  dict\_values([3.4, 8.8, 25.0, 17.3]) | |

**Exemplo:**

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

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

**9.2Exercícios:dicionários**

**1 –** Dada a tabela a seguir, crie um dicionário que a represente:

|  |  |
| --- | --- |
| **Lanchonete** | |
| **Produtos** | **Preços R$** |
| Salgado | R$ 4.50 |
| Lanche | R$ 6.50 |
| Suco | R$ 3.00 |
| Refrigerante | R$ 3.50 |
| Doce | R$ 1.00 |

|  |
| --- |
| **2 –** Considere um dicionário com 5 nomes de alunos e suas notas. Escreva um programa que calcule a média dessas notas. |

**11. FUNÇÕES**

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

**11.1 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 parâmetrosé opcional.

**Exemplo:** Função simples

**Defhello**():

**print**("Olá Mundo!!!")

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

>>>hello()

**Olá Mundo!!!**

**11.2 Parâmetros e argumentos**

Parâmetros são as variáveis que podem ser incluídas nos parênteses das funções. Quando a função é chamada são passados valores paraessas 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 imprimiro maior entre 2 valores

**Defmaior**(x,y):

**If**x>y:

**print**(x)

**else:**

**print**(y)

>>>maior(4,7)

**11.3 Escopo das variáveis**

Todavariável utilizadadentro de uma funçãotem 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 declarara 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.

**11.4Retorno de valores**

O comando**return**é usado para retornar um valor de uma funçãoe encerrá-la. Caso não seja declarado um valor de retorno, a função retornaovalor **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.

**11.5 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:**

**defcalcula\_juros**(valor, taxa=10):

juros = valor\*taxa/100

**return**juros

>>>calcula\_juros(500)

**50.0**

**11.6Exercícios: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 qualquertipo. A função deve imprimirtodos oselementosda lista numerando-os.

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