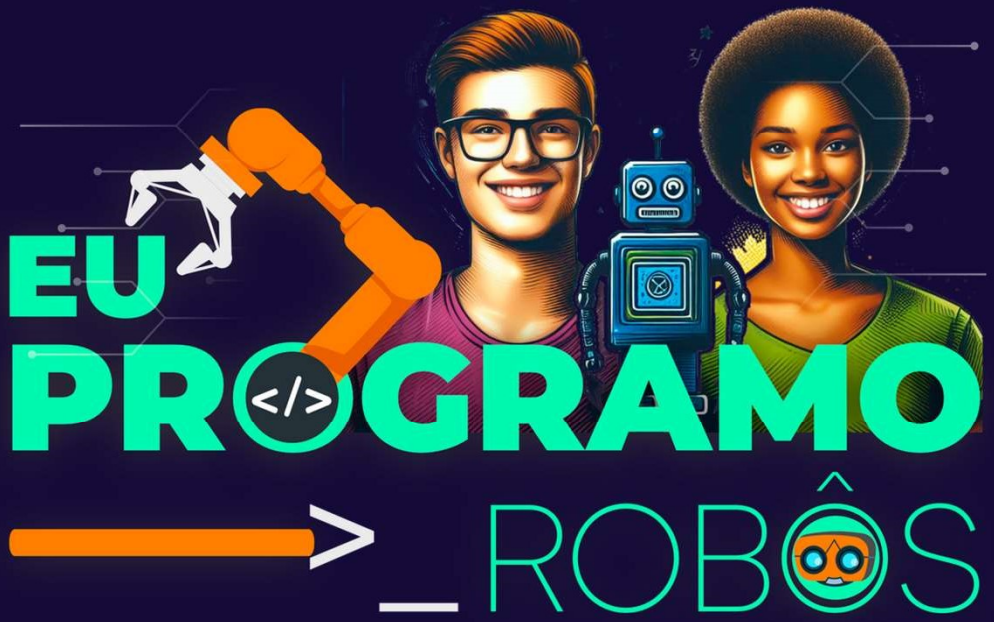
Robótica II

Prof. André Nasserala
andre.nasserala@ufac.br



INSTITUTO FEDERAL Acre · FACTO · PPI4.0 · cits

Linguagem de Programação Python

Introdução ao Python

- Python é uma linguagem de programação multiplataforma que permite desenvolver aplicações para games, desktops, web e dispositivos móveis.

- Além disso, ela pode se comunicar com outras aplicações que foram desenvolvidas em outras linguagens como C, C++, Java e C#.

Antes.... Um pouco de VIM

- O Vim (Vi IMproved) é um editor de texto modal altamente personalizável e popular entre desenvolvedores e usuários experientes.
- Ele é conhecido por sua curva de aprendizado íngreme, mas oferece poder e flexibilidade incomparáveis para edição de texto, programação e tarefas complexas.
- Para instalar:
- **sudo apt-get install vim -y**

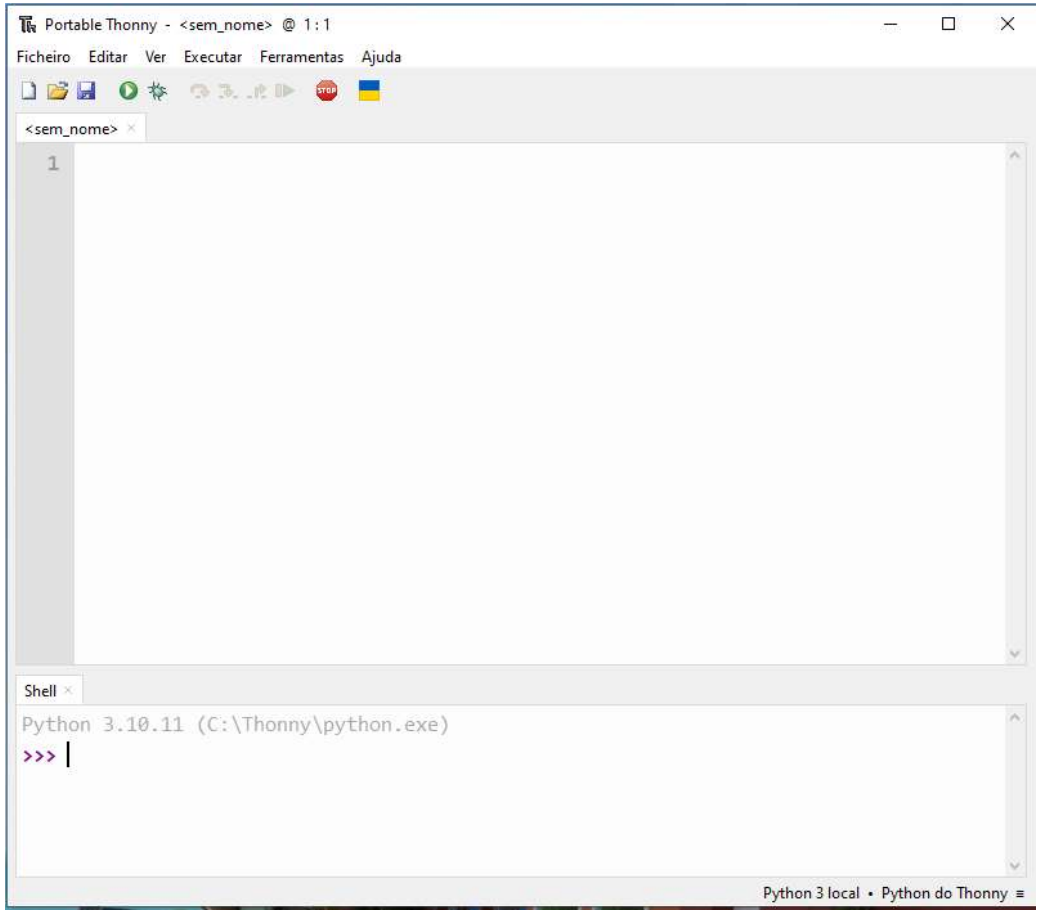
```
nasserala@pinasser:~ $ sudo apt-get install vim -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Suggested packages:
  ctags vim-doc vim-scripts
The following NEW packages will be installed:
  vim
0 upgraded, 1 newly installed, 0 to remove and 1 not
Need to get 0 B/1,424 kB of archives.
After this operation, 3,875 kB of additional disk sp
Selecting previously unselected package vim.
(Reading database ... 60364 files and directories cu
Preparing to unpack .../vim_2%3a9.0.1378-2_arm64.deb
Unpacking vim (2:9.0.1378-2) ...
Setting up vim (2:9.0.1378-2) ...
update-alternatives: using /usr/bin/vim.basic to pro
update-alternatives: using /usr/bin/vim.basic to pro
update-alternatives: using /usr/bin/vim.basic to pro
update-alternatives: using /usr/bin/vim.basic to pro
update-alternatives: using /usr/bin/vim.basic to pro
update-alternatives: using /usr/bin/vim.basic to pro
update-alternatives: using /usr/bin/vim.basic to pro
nasserala@pinasser:~ $
```

Antes.... Um pouco de VIM

- Usando: **vim arquivo.py**
- Ao abrir um arquivo, como anteriormente, devemos abrir o modo de inserção apertando a tecla “i”(aparecerá “INSERT” no rodapé);
- Agora podemos digitar normalmente, e quando terminarmos de digitar devemos sair do modo de inserção apertando a tecla ESC;
- Fora do modo de inserção podemos salvar o arquivo com os comandos:
- **ESC + :x**
- Esse comando sai salvando o arquivo, se quiser sair e ignorar o que foi feito use **ESC + :q!**

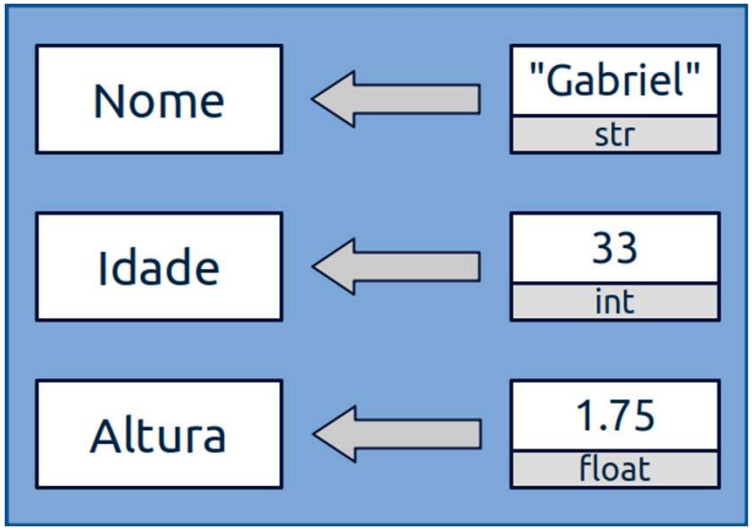
Antes.... Um Thonny

- Thonny é um ambiente de desenvolvimento integrado (IDE) especialmente projetado para quem está começando a programar em Python.
- Sua interface intuitiva e recursos simplificados tornam o aprendizado da linguagem muito mais fácil e agradável.



Conceito de Variável em Python

- O conceito de variável em Python é representado sempre por um objeto (tudo é objeto), e toda variável é uma referência.
- Na maioria das linguagens de programação, quando iniciamos uma variável e atribuímos um valor a ela, essas carregam valores que são alocados em memória, e quando alteramos os seus valores, estamos alterando o valor na memória também.
- Porém, no Python as variáveis armazenam endereços de memória e não os valores.



Conceito de Variável em Python

- Outro ponto a considerar é que em Python uma variável não tem um tipo fixo, mas sim apenas o tipo do conteúdo, como mostra o exemplo01.

```
x = [1,2,3]
y = x
x.append(4)
print(y)
```

exemplo01.py

- Na linha 1 é criada uma variável x que recebeu um vetor de inteiros.
- Em seguida, foi referenciada uma variável y onde é atribuída a referência de x.

- Para executar:

$ python3 exemplo01.py

Conceito de Variável em Python

- Na linha 3 observamos que x também é um objeto quando invocamos o método append, que adiciona um elemento ao vetor x.
- Na linha 4 utilizamos o método print() para exibir a variável y.
- Em Python, uma variável não tem um tipo fixo, apenas o tipo do conteúdo atual, por isso o vetor se atualizou e assim o valor de y.
- Em relação ao nome que podemos atribuir as variáveis, pode-se utilizar letras, o caractere de sublinhado _ e números.
- Os comentários em Python são representados pelo símbolo # e colocado em cada linha.

Tipos de Dados em Python

- Uma característica que facilita muito a vida do desenvolvedor são as conversões de tipos em Python.
- Veja alguns exemplos abaixo:

```
a = float(22/5)
b = int(4.5)
c = int(3.9)
d = int(0xff563)
e = float(int(3.9))
f = int(float(3.9))
g = int(float(3))
h = round(3.9)
i = round(3)
j = int(round(3.9))
print(a,"\n",b,c,d,e,f,g,h,i,j)
```

exemplo02.py

Tipos de Dados em Python: Inteiros

- Para a declaração de números inteiros, necessitamos que estes estejam entre -2147483648 a 2147483647.
- Cada um ocupa quatro bytes na memória e pode armazenar tantos valores decimais (de base 10), quanto valores octais (de base 8) e hexadecimais (base 16).
- Para declarar um inteiro octal, o número 0o (zero, “o”) tem que ser prefixado ao número, como em 0o123; e para definir um número hexadecimal, o prefixo 0x ou 0X deve ser utilizado, como 0xFFFFFF ou 0X006699.
- Observe o exemplo03.py

```
a = 42 # numero decimal decimal
b = 0o10 # numero octal
c = 0xA # hexadecimal

print("Valor de a:",a)
print("Valor de b:",b)
print("Valor de c:",c)
```

Tipos de Dados em Python: Long

- Representa números inteiros longos e pode armazenar números tão grandes quanto a memória puder armazenar.
- Assim como o tipo int, o long também pode armazenar números tanto decimais, quanto octais e hexadecimais.
- Para declarar um valor long mesmo se o número a ser atribuído estiver na faixa de valores do tipo int é necessário sufixar a letra L (minúscula ou maiúscula) como em 524511574362l, 0xDDEFBDAEFBDAECBFBAEL e 0122L.
- Exemplo:
- a = 0xDDEFBDAEFBDAECBFBAEL
- Representa: hexadecimal Long

Tipos de Dados em Python: Float

- Representa números reais e que possuem sinal de expoente (e ou E).
- Esses números são comumente chamados de floating-point numbers ou números de ponto flutuante.

- Por exemplo:
  - 0.0042
  - .005
  - 1.14159265
  - 6.02e23 (o mesmo que 6.02×10^{23})

Tipos de Dados em Python: Bool e NoneType

- O tipo bool foi adicionado na versão 2.2 do Python como uma especialização do tipo int.
- Os valores do tipo bool podem representar dois valores completamente distintos: True (igual ao int 1) e False (igual ao int 0) para, respectivamente, verdadeiro e falso.
- Exemplo: a = True e b = False.

- NoneType é o tipo de None, uma constante embutida do Python que, assim como True e False, e é frequentemente utilizada para representar a ausência de um valor, similar ao null na linguagem C e derivadas.
- Exemplo: a = None (o mesmo que null em Java).

Tipos de Dados em Python: String

- Para atribuirmos a uma variável uma referência do tipo string, basta que coloquemos entre aspas simples, duplas ou triplas, como mostra o exemplo04.py

```
a = 'Isso é uma String com aspas Simples'
b = "Isso é uma String com aspas Duplas"
c = """Isso é Uma String com aspas Triplas"""
print(a + "\n" + b + "\n" + c)
print(a, "\n" , b , "\n" , c)
print(a + b + c)
```

exemplo04.py

- Este código gera uma saída com cada frase em uma linha nova, mesmo estando concatenada.
- Isso acontece porque foi utilizado o recurso de quebra de linha com o \n.

Tipos de Dados em Python: String

- Em Python, tudo é objeto, assim as Strings são objetos que tem embutidos vários métodos.
- Vamos analisar o seguinte código do exemplo05.py.

```
#coding: utf-8
meu_nome = "Sintaxes Python"
meu_nick = 'Devmedia'
print ("Nome: %s, Nick: %s" % (meu_nome.upper(), meu_nick))
print ("Meu nome comeca com a letra ", meu_nome[0])
print ("Meu nome comeca com a letra ", meu_nome[0].lower())
print ("Meu primeiro nome é ", meu_nome[0:7])
```

exemplo05.py

- Na linha 1 é utilizado um comentário #coding: utf-8, que permite a acentuação.

Tipos de Dados em Python: String

- Nas linhas 2 e 3 são criadas variáveis as quais atribuímos uma string.
- Na linha 4 utilizamos um recurso muito conhecido em C, no qual exibimos uma string e indicamos quais os tipos de variáveis que serão utilizadas naquela referência, no caso de string utiliza-se o %s.
- Note que a exibição é respectiva a sequência da declaração na mesma linha.
- Use o método upper() para deixar todas as letras em caixa alta.
- Na linha 5 exibimos novamente a concatenação de uma string com a variável que é um objeto, contudo, sabemos que string são vetores de caracteres e utilizando a variável seguida de um índice o compilador mostra apenas a primeira posição do vetor.

Tipos de Dados em Python: String

- Na linha 6 esse mesmo vetor ainda invoca o método lower() que mostra o caractere em caixa baixa, e na linha 7 esse vetor é exibido da sua posição 0 até a posição 7.
- Sobre a linha 4 na Tabela temos os tipos de variáveis para os formatos dos símbolos e suas respectivas conversões.

Formato do Símbolo	Conversão
%c	Caractere
%s	String convertida pelo método str() tem prioridade
%i	Sinal decimal inteiro
%d	Sinal decimal inteiro
%u	Decimal inteiro sem negativos (unsigned)
%o	Octal inteiro
%x	Hexadecimal Inteiro (letras em caixa baixa)
%X	Hexadecimal Inteiro (Letras em Caixa ALTA)
%e	Notação exponencial (Para letras minúsculas 'e')
%E	Notação exponencial (Para letras MAIÚSCULAS 'E')
%f	Ponto flutuante (Números reais)
%g	O mais curto de 'f%' e '%e'
%G	O mais curto de 'f%' e '%E'

Tipos de Dados em Python: String

- Outros métodos interessantes são o capitalize(), que faz com que apenas o primeiro caractere seja exibido em caixa alta.

- O método strip() retorna uma cópia da string com os espaços em branco antes e depois removidos.

- O método startswith(prefixo) retorna True se uma string começa com o prefixo passado por argumento, e False caso contrário.

Operadores

- A Tabela resume como são utilizados os operadores na linguagem:

Aritméticos	Comparação	Lógicos
+	==	and
-	!=	or
*	>	not
/ ou // (parte inteira)	<	
%	>=	
+= -= *= /=	<=	
**	in in not	
is		

Entrada de Dados no Python

- Em Python, a leitura de dados do teclado é feita através das funções sys.stdin.readline() e input, como mostra o exemplo06.

```
import sys

nome = input("Digite seu nome: ")
idade = input("Digite sua idade: ")
print("Digite seu sexo: ")
sexo = sys.stdin.readline()

print("Nome:"+nome + "\n" + "Sexo: %s Idade: %s" %(sexo,idade))
```

exemplo06.py

Entrada de Dados no Python

- Na linha 8 são utilizadas duas maneiras para exibir as strings, e também são utilizadas duas maneiras de fazer a leitura do teclado:
  - uma diretamente na variável com exibição de uma mensagem,
  - e outra em que utilizamos um método readline(), que está contido na importação feita na linha 1.

Comandos de Decisão

- Para quem já tem contato com alguma linguagem de programação, é possível notar que é muito simples os comandos condicionais de Python:

```
if CONDIÇÃO :
    BLOCO DE CÓDIGO
elif CONDIÇÃO :
    BLOCO DE CÓDIGO
else :
    BLOCO DE CÓDIGO
```

IF, ELIF, ELSE no Python

- Uma observação que deve ser feita é que em Python não usamos as chaves {} para iniciar e finalizar um bloco de instruções.
- Para que você delimite onde cada bloco começa e acaba, basta identar o código, como mostra a exemplo07.

```
# coding:utf-8
dedos = int(input("Você tem quantos anos? "))

if dedos == 18:
  print("Você tem 18 anos")
elif dedos > 18:
  print("Você tem mais de 18 anos")
else:
  print("Você é menor de idade")
```

exemplo07.py

Cadeias de IF

- Outra forma de utilizar o if, elif e else, é implementando-o como uma estrutura, tipo switch, onde cada elif vai ter um case dentro do switch, como mostra o exemplo da exemplo08.

```
# coding: utf-8
var1 = int(input("Digite um Número para var1: "))
var2 = int(input("Digite um Número para var2: "))
if var1 == 1:
  print("Número var1 igual a 1")
elif var1 == 2 or var2 == 3:
  print("var1 diferente de 1 ou var2 diferente de 2")
elif var1 >= 1000 or var2 <= -1000:
  print("var1 maior que 1000 ou var2 menor que -1000")
else:
  print("nenhuma das alternativas anteriores")
```

exemplo08.py

Função RANGE()

- Existe também a função range(), que retorna um array, ou seja, os elementos são representados em uma sequência, como mostra o exemplo09.

```
print(*range(10)) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(*range(5, 10)) # [5, 6, 7, 8, 9]
print(*range(10, 0)) # [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
i=int(input("Digite um Número: "))
if i in range(0,10):
    print("Está Contido")
else:
    print("Não está Contido")
```

exemplo09.py

- Nas linhas 1 a 3 é impresso na tela os valores (que estão em comentários), e a partir da linha 4 é verificado se o valor digitado pelo usuário está contido na sequência de 0 a 9.

Laço de Repetição FOR

- No laço de repetição for é implementado em Python e pode ser utilizado de diversas maneiras, como exibido no exemplo10.

```
#coding: utf-8
for fruta in [ "banana", "maca ", "uva" ]:
   print ("Fruta : " + fruta)
print("------------------------------")
for i in range (0,10): #(Inclui o Zero, NÃO INCLUI O 10)
   print ("i = " + str ( i ))
print("------------------------------")
for i in range(0,10):
   print ('Não realizado1')
   if (i == 0):
      break
print ('Realizado')
```

exemplo10.py

Laço de Repetição FOR

```
#coding: utf-8
for fruta in [ "banana", "maca ", "uva" ]:
   print ("Fruta : " + fruta)
print("--------------------------------")
for i in range (0,10): #(Inclui o Zero, NÃO INCLUI O 10)
   print ("i = " + str ( i ))
print("--------------------------------")
for i in range(0,10):
   print ('Não realizado1')
   if (i == 0):
      break
print ('Realizado')
```

- Nesse exemplo, o for da linha 2 armazena três valores para a variável fruta.
- No for da linha 5 armazena os valores da sequência range para a variável i e dentro do for da linha 8 demonstra que é possível colocar estruturas de condições.

Laços de repetição WHILE no Python

- O laço de repetição while no Python tem basicamente a mesma estrutura do for, porém, como no for ou if, não se utilizam os parênteses para definir a condição e seu bloco de instrução também é definido com a identação, como mostra o exemplo11.

- A instrução pass faz nada, ou seja, se quisermos definir um ciclo ou função que não tenha nenhuma instrução a ser executada, usamos o pass.

```
i = 0
while i < 10:
    print ("i = ",i)
    i += 1

while True:
    pass
```

exemplo11.py

Exceções e Listas

- As exceções podem ser tratadas utilizando o recurso do try...catch, que tem como sintaxe o código do exemplo12.
- Uma lista é um conjunto de valores indexados por um número inteiro chamado de índice, que se inicia em zero.
- Os elementos contidos em uma lista podem ser de qualquer tipo, até mesmo outras listas, e não precisam ser todos do mesmo tipo.
- Uma lista é delimitada por colchetes e seus elementos separados por vírgulas, como vemos no exemplo13.

```
try:
  print (1 + 'st try')
except:
   print ("Isto é uma exceção 1")

a = 5
b = 0

try:
  print (int(a/b))
except:
   print ("Isto é uma exceção 2")
```

exemplo12.py

Exceções e Listas

- Utilizamos um método que recebe uma lista por parâmetro e devolve o tamanho da lista.
- Note que a função range() também existe no laço for.
- Na linha 4 é exibido o item 0 da lista.
- Veja no exemplo14 execução e o resultado de como acessar o conteúdo das listas em Python.
- Repare que ao usar índices negativos, as posições são acessadas a partir do final da lista.

```
lista = [1, 2, 3]
print ("Tamanho de lista:", len(lista))

lista[0]=9
print("-----------")
for i in range(len(lista)):
    print (lista[i])
```

exemplo13.py

Exceções e Listas

- Linha 2 imprime 1;
- Linha 4 imprime valor 3;
- Linha 5 imprime 2;
- Linha 6 imprime [1,2];
- Linha 7 imprime [1,2];
- Linha 8 imprime [2,3].

```
lista = [1, 2, 3]
print (lista[0])
print (lista[1])
print (lista[-1])
print (lista[-2])
print (lista[0:2])
print (lista[:2])
print (lista[1:])
```

exemplo14.py

Exceções e Listas

- A função capitalize() deixa a string em caixa alta.
- No exemplo15 essa função está sendo aplicada em uma lista e dentro de um laço de repetição.
- A função remove() tira um item de uma lista em Python, como mostra o exemplo no exemplo16.

```
frutas=['laranja','banana','abacaxi']

frutas.sort()
print (frutas)
frutas[0]='framboesa';

for item in frutas:
  print (item.capitalize())
```

exemplo15.py

Exceções e Listas

- O código é responsável por criar uma lista adicionando elementos, exibir os convidados e remover um deles, exibindo logo após uma lista atualizada dentro do laço for.

```
convidados = [] #define uma lista vazia
convidados.append ('Zeze')
convidados.append ('Luciano')
convidados.append ('Bruno')
convidados.append ('Marrone')
print ("Tenho ", len(convidados)," Convidados")
convidados.sort()
print ("Sao eles:")
print (convidados)
print ("O primeiro convidado eh o ",convidados[0])
convidados.remove("Marrone"); #aqui tiramos o Exemplo 4 da lista
print ("Agora tenho somente ", len(convidados)," Convidados")
print ("Sao eles:")
for convidado in convidados:
  print (convidado)
```

exemplo16.py

Tuplas e Dicionários

- Até agora foram vistos dois tipos compostos: strings, que são compostos de caracteres; e listas, que são compostas de elementos de qualquer tipo.
- Uma das diferenças que notamos é que os elementos de uma lista podem ser modificados, mas os caracteres em uma string não.
- Em outras palavras, strings são imutáveis e listas são mutáveis.
- Há outro tipo em Python chamado tupla (tuple), que é similar a uma lista, exceto por ser imutável.
- Sintaticamente, uma tupla é uma lista de valores separados por vírgulas e com aspas simples, como vemos no exemplo17.

Tuplas e Dicionários

- Os dicionários em Python são estruturas de dados que permitem “traduzir” uma chave para um valor, ou seja, são estruturas de dados que implementam mapeamentos.
- Um mapeamento é uma coleção de associações entre pares de valores, onde o primeiro elemento do par é chamado de chave e o outro de conteúdo.

```
frutas=['laranja','banana','abacaxi']

frutas.sort()
print (frutas)
frutas[0]='framboesa';
for item in frutas:
  print (item.capitalize())
```

exemplo17.py

Tuplas e Dicionários

- De certa forma, o mapeamento é uma generalização utilizada para acessar os dados das listas pelas suas chaves que funcionariam como índices, contudo estas chaves necessitam ser imutáveis. Vejamos o exemplo18.
- Esse código representa as declarações de listas com atributos de um carro, como marca. A variável dicionário recebe as chaves mapeadas e depois imprime a que está na primeira posição.

```
veiculo = {}
veiculo['marca'] = 'Puma'
veiculo['modelo'] = 'GTB'
veiculo['ano'] = 1978
print (veiculo['marca'])
dicionario = {"chave":"17,532","chave2": "17,365" }
print (dicionario ["chave"]) #imprime valor (17,532)
```

exemplo18.py

Matriz com Listas no Python

- Em Python o conceito de arrays multidimensionais não existe.
- O que existe é uma alternativa, que são os arrays encadeados, ou seja, cada elemento do array corresponde a outro array, ou como uma list de list em Java.
- Veja no exemplo19, como pode ser criada uma matriz de 10 linhas e cinco colunas e como ela pode ser “varrida”.

```
#Criando uma matriz de 10x5 inicializada com 0
lin=10
col=5
matriz=[]
for i in range(0,lin):
  linha=[]
  for j in range(0,col):
    linha.append(j)
  matriz.append(linha)
#varrendo a matriz
for i in range(0,len(matriz)):
  for j in range(0,len(matriz[0])):
    print (matriz[i][j], end=' ')
  print()
```

exemplo19.py

Matriz com Listas no Python

- O primeiro for (onde as linhas são adicionadas, logo após terminar de percorrer as colunas em uma variável que é criada localmente) é definido especificando o começo incluso e a linha não inclusa, pois um vetor inicializa na posição zero.

- No for mais interno do primeiro exemplo nota-se o mesmo uso, onde este adiciona zeros nas colunas, retorna ao laço mais externo e adiciona mais uma linha.

Funções

- Assim como em outras linguagens, uma função pode receber alguns parâmetros e devolver um resultado.
- A sintaxe inicia-se com a palavra def, seguida do nome da função e dos parênteses que podem conter seus parâmetros.
- As funções também são consideradas objetos e têm alguns atributos, dentre os quais os mais úteis são __doc__, o qual contém a documentação da função, e __name__, que contém o nome (apelido) da função.
- Mas antes de entrar diretamente no código, é importante abordar sobre a recursividade da linguagem e sobre a documentação da função.
- Recursividade de uma função basicamente é a definição de uma sub-rotina, seja ela função ou método, que tem a capacidade de invocar a si mesmo como, por exemplo, o fatorial.

Funções

- Outro recurso é a documentação, que nada mais é do que um comentário dentro da função que é exibido quando utiliza-se o método __doc__, como vemos no caso do exemplo20.

```
def fatorial (numero): #função recursiva
    """
    Funcao recursiva
    As três aspas duplas é a documentação
    """
    if numero <= 1:
      return 1
    else:
      return (numero * fatorial (numero - 1 ))
for n in range ( 1, 11 ):
    print ("Fatorial de",n," eh ", fatorial(n))
print ("\nDoc. da função:\n"+fatorial.__doc__)
```

exemplo20.py

Funções

- A função retorna o valor do fatorial do número que está dentro do for.
- A linha 12 imprime a documentação que está dentro da função.
- Também é possível utilizar função em Python com parâmetros opcionais como, por exemplo, uma função que calcula a potência, onde poderíamos ou não informar o expoente.

- Caso não fosse informado assumira o padrão de 2, como no código do exemplo21.

Funções

- Caso não seja informado o expoente, será considerado o expoente 2. A linha 10 imprime valor 4 e a linha 11 imprime valor 256.

```
def potencia(base,exp=2): # função com parâmetro opcional
    if exp==0:
        return 1
    pot=base;i=1;
    while i < exp:
        pot=pot*base
        i=i+1
    return pot
print("Começa aqui!")
print (potencia(2)) # imprime 4
print (potencia(2,8)) #imprime 256
```

exemplo21.py

Arquivos no Python

- Manipular arquivos em Python é extremamente simples, como vemos no exemplo22.

```
arq=open('meuarquivo.txt', 'w')
arq.write('gravando em um arquivo é simples')
arq.close ()
```

exemplo22.py

- Um arquivo aberto é um objeto instanciado pela função open().
- Repare que na linha 1 é passado por parâmetro o local do arquivo (foi utilizado o disco D: para simplificar o exemplo).
- Dependendo da configuração pode dar um erro referente aos privilégios de gravação, mas se isso ocorrer, utilize uma pasta comum do usuário logado.
- Como segundo argumento é passado o ‘w’, que indica a criação de um novo arquivo para a escrita.

Arquivos no Python

- Na Tabela vemos a lista de argumentos que podemos usar:

Argumento	Ação
W	Sempre cria um novo arquivo para escrita.
w+	O mesmo que “w”, mas abre para escrita e leitura.
R	Abre para leitura.
A	Abre o arquivo para escrita, mantendo o conteúdo e posiciona o cursor no final do arquivo, utilizado para adicionar conteúdos.
a+	O mesmo que ‘a’, mas abre para leitura e escrita.
B	Em combinação com os modos acima, manipula os arquivos binários (wb, rb, ab+,...).

Arquivos no Python

- A classe utilizada para executar as operações é a file, e assim que é executado o comando open é retornado um objeto do tipo file.
- No exemplo22 foram vistos três exemplos de métodos possíveis com arquivo: o open, write (escreve no arquivo) e close (que fecha o arquivo).
- Para leitura do arquivo existe o método read, e se utilizarmos somente, ele nos retorna todo o conteúdo do arquivo, como podemos ver na implementação do código do exemplo23.

```
arq = open('meuarquivo.txt') # r é default
leitura = arq.read(3)
print(leitura)
restante=arq.read()
print(restante)
```

exemplo23.py

Arquivos no Python

- É aberto o arquivo para leitura e é passado por parâmetro a quantidade de caracteres que se deseja ler.
- Se for utilizado novamente sem parâmetro, ele retorna o restante do texto não lido.

- A linguagem de programação Python é realmente muito simples, fácil e são utilizadas poucas linhas para se trabalhar, facilitando no desenvolvimento e em seu desempenho.

Bibliografia

- MQTT. Web: https://mqtt.org/. Pesquisado em Abril de 2024.
- DEVMEDIA, Igor. Artigo Python Tutorial. Disponível em: https://www.devmedia.com.br/python-tutorial/33274.
- MALVINO, Albert Paul. Eletrônica: Volume 1. 4.ed. São Paulo – SP: Makron Books, 1997. ISBN: 8534603782.
- SENAI, Senai SP. FUNDAMENTOS DE ELETRONICA - 1ªED. Editora: Senai SP – São Paulo 2015. ISBN: 9788583932086
- - WILSON, J. A. e Milton Kaufman. Eletrônica Básica - Teoria e Prática - Volume 2. São Paulo: Editora: Rideel, 1980.
- - PEREZ, Anderson Luiz Fernandes, Heron Pereira, Cristiano Pereira de Abreu, Renan Rocha Darós. Oficina de Robótica. UFSC – Programação Básica em Arduino - 2015. Disponível em: http://oficinaderobotica.ufsc.br/programacao-basica-em-arduino/.