Aula 2 - Introdução ao Python

AGENDA

- A linguagem Python
- Organização do código
- Tipagem dinâmica
- Casting
- Laços de repetição e condicionais
- Estrutura de dados
- Declaração de funções
- Entrada e Saída

A Linguagem Python

- Linguagem de alto nível Criada por Guido van Rossum em 1991
- Características principais:
  - Linguagem interpretada de *scripts*
  - Multiparadigma
    - Orientação à objetos
    - Funcional
    - Procedural
  - Tipagem dinâmica e forte
- Versão atual: Python 3.9

Organização do código

- Diferentemente de outras linguagens a organização do código escrito em Python faz uso da indentação para organização de blocos de código

```c
int main(){
    int i = 0;
    while(i<20){
        if(x%2 == 0){
            printf("%d",i);
            printf("É par");
        }
        else{
            printf("%d",i);
            printf("É impar");
        }
        i++;
    }
}
```

```python
i = 0
while(i<20)
    if(i%2==0)
        print(i)
        print("É par")
    else
        print(i)
        print("É impar")

    i=+1
```

Tipagem Dinâmica

- A linguagem determina em tempo de execução o tipo da variável de acordo com a variável atribuída à mesma.

```
[1] a = 'Hello World'
    type(a)

    str
```
String

```
[2] a = 2021
    type(a)

    int
```
Inteiro

```
[3] a = 74.05
    type(a)

    float
```
Float

```
[4] a = 5 + 3j
    type(a)

    complex
```
Complexo

```
[5] a = 10 > 2
    type(a)

    bool
```
Booleano

Casting

- Mesmo com a tipagem dinâmica, em certos casos queremos especificar de forma explícita o tipo que variável deve assumir, isso é feito por meio do *CASTING*
- Exemplos de *casting*

```
[9]  i = int(2021)
     print(i)
     i = int(2021.97)
     print(i)
     i = int("2021")
     print(i)
```

```
2021
2021
2021
```

```
[10] i = float(2021)
     print(i)
     i = float(2021.97)
     print(i)
     i = float("2021.999888")
     print(i)
```

```
2021.0
2021.97
2021.999888
```

```
[12] i = str(2021)
     print(i)
     i = str(2021.97)
     print(i)
     i = str(4 + 2j)
     print(i)
```

```
2021
2021.97
(4+2j)
```

Condicional

- Blocos condicionais em Python são escritos utilizando *if-else-elif*
- *elif* é equivalente ao *else if*

Sintaxe básica SE:

```
1  x = 10
2  if x > 0:
3      #Do something
```

Sintaxe básica SE-SENÃO:

```
1  x = 10
2  if x > 0:
3      #Do something
4  else:
5      #Do another thing
6
```

Sintaxe básica SE-SENÃO SE:

```
1  x = 10
2  if x > 5:
3      #Do something
4  elif x > 2:
5      #Do another thing
6  else:
7      #Do another thing
8
```

Operadores Lógicos e Relacionais

Relacionais:

==	Igual: a == b
!=	Diferente: a != b
>	Maior: a > b
<	Menor: a < b
>=	Maior/igual: a >= b
<=	Menor/igual a <= b

Lógicos

and	True and True retorna True False and True retorna False False and False retorna False
or	True or True retorna True False or True retorna True False or False retorna False
not	not True retorna False not False retorna True

Operadores Lógicos e Relacionais

Relacionais:

```
[3]  x = 10
     y = 15
     if (x == y):
       print('Igual')
```

```
[4]  if (x != y):
       print('Diferente')

     Diferente
```

```
[5]  if (x > y):
       print("Maior")
```

```
[6]  if (x < y):
       print("Menor")

     Menor
```

```
[7]  if (x <= y):
       print("Menor/Igual")

     Menor/Igual
```

```
[9]  if (x >= y):
       print("Maior/Igual")
```

Or:

```
[13] if (x != y) or (x < y):
       print("Verdadeiro")

     Verdadeiro
```

```
[15] if (x != y) or (x > y):
       print("Verdadeiro")

     Verdadeiro
```

```
[17] if (x == y) or (x > y):
       print('Falso')
```

Not:

```
[21] if not (x > y):
       print("Verdadeiro")

     Verdadeiro
```

```
[23] if not (x != y):
       print('Falso')
```

And:

```
[18] if (x != y) and (x < y):
       print("Verdadeiro")

     Verdadeiro
```

```
[19] if (x != y) and (x > y):
       print("Falso")
```

```
[20] if (x == y) and (x > y):
       print('Falso')
```

Laços de Repetição

- Para a escrita de laços de repetição o Python permite o uso de *for* ou *while*
- Laço *for* - Itera sobre os itens de uma lista
- Laço *while* - executa um bloco de código enquanto a condição determinada for verdadeira

Sintaxe básica do *for*:

```
1  for item in lista:
2      print(item)
3
```

Sintaxe básica do *while*:

```
1  while (condiçao):
2      #Do something
3
```

Laços de Repetição - Exemplos

- A função range(start, stop, step) retorna uma lista de números
    - range(5) - números de zero a quatro
    - range(2,12) - números de dois a onze
    - range(0,12,2) - números de zero a dez de dois em dois

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

```
0
1
2
3
4
```

```
[27] for i in range(2,12):
       print(i)
```

```
2
3
4
5
6
7
8
9
10
11
```

```
[28] for i in range(0,12,2):
       print(i)
```

```
0
2
4
6
8
10
```

```
[25] lista = ['A','B','C','D','E','F']

     for i in lista:
       print(i)
```

```
A
B
C
D
E
F
```

Estruturas de Dados - Listas

- Estrutura de dados mais básica presente no Python
- Armazena dados em sequência onde cada um é identificado por um índice
- Armazena qualquer tipo de dado primitivo(string, float, int, etc....)
- Permite o armazenamento de dados de diferentes tipos em uma mesma lista
- Permite a inserção de listas dentro de listas

```
[8] lista = [1,'A',3.14] #declaração
    print(lista)

    [1, 'A', 3.14]
```

```
[9] lista.append(10)#adiciona um elemento ao fim da lista
    print(lista)

    [1, 'A', 3.14, 10]
```

```
[10] lista.remove('A')#remove o primeiro elemento igual ao argumento da função
     print(lista)

     [1, 3.14, 10]
```

```
[11] lista.insert(1,64)#insere o elemento especificado em uma dada posição, neste caso na posiçao 1
     print(lista)

     [1, 64, 3.14, 10]
```

```
[12] lista.pop(2)#remove o elemento armazenado na posição especificada
     print(lista)

     [1, 64, 10]
```

Estruturas de Dados - Sets

- Coleção de itens desordenada que não permite elementos duplicados
- Permite operações de diferença simétrica, interseção e união

```
[13] conjuntoA = {1,2,3,4,5}
     conjuntoB = {4,5,6,7,8}
     print(conjuntoA.union(conjuntoB))

     {1, 2, 3, 4, 5, 6, 7, 8}
```

```
[15] print(conjuntoA.intersection(conjuntoB))

     {4, 5}
```

```
[16] print(conjuntoA - conjuntoB)

     {1, 2, 3}
```

Estruturas de Dados - Dicionários

- Coleção desordenada de itens do tipo chave:valor
- Exemplo de item (<key>,<value>): (200,"João")
- Chave e valor podem ser de qualquer tipo primitivo, contanto que a chave seja única

```
[17] dicionario = {1:'Alice',2:'Bruno',3:'José',4:'Alex'}
     print(dicionario)

     {1: 'Alice', 2: 'Bruno', 3: 'José', 4: 'Alex'}
```

```
[18] dicionario[5] = 'Maria'
     print(dicionario)

     {1: 'Alice', 2: 'Bruno', 3: 'José', 4: 'Alex', 5: 'Maria'}
```

Declaração de funções

- Sintaxe básica: *def* <Nome da função>*(parâmetros)*
- Diferentemente de outras linguagens não é necessária a especificação do tipo da função e de seus parâmetros
- Permite mais de um valor de retorno

```
[22] def CalculaIMC(altura,massa):
       imc = massa / (altura*altura)
       return imc

     imc = CalculaIMC(1.83,90)
     print(imc)
```

```
26.874496103198062
```

```
def AvaliaCirculo(raio):
  perimetro = 2*3.14*raio
  area = 3.14*raio*raio
  return perimetro, area

perimetro, area = AvaliaCirculo(3)

print(perimetro)
print(area)
```

```
18.84
28.259999999999998
```

Entrada e Saída

- As operações de entrada e saída no Python são executadas por meio das funções *input* e *print*
- A função *input* retorna o valor inserido pelo usuário no formato de *string*
- A função *print* permite a concatenação de valores separados por vírgula

```
[28] nome = input()

     José da Silva

[29] print(nome)

     José da Silva

[30] print(nome,'tem 23 anos')

     José da Silva tem 23 anos
```

GRATIDÃO!

UNIFESP

UNIVERSIDADE FEDERAL DE SÃO PAULO

1933