CAPÍTULO – 04 PYTHON

FUNÇÕES

FUNÇÃO ⇒ sequência de comandos que executam uma tarefa.
⇒ funciona como uma unidade de execução

```
>>> type(32)
<class 'int'>
```

FUNÇÃO
ARGUMENTO ⇒ valor ou variável que é passado como entrada para a função.

→ RECEBE ⇒ argumento
→ RETORNA ⇒ resultado

* BUILT-IN FUNCTIONS (FUNÇÕES INTEGRADAS)

→ São funções criadas dentro do python para resolver problemas comuns.

Palavras reservadas

```
>>> max('Hello word')
'w'
```
→ maior caractere do argumento

```
>>> min('Hello word')
' '
```
→ menor caractere do argumento

```
>>> len('Hello word')
11
```

* FUNÇÕES DE CONVERSÃO DE TIPOS

```
>>> int('32')
32
>>> int('ola')
ERRO
```

CAPÍTULO - 04

```
>>> int(3.99999)
3
>>> int(-2.3)
-2
```

```
>>> float(32)
32.0
>>> float('3.14159')
3.14159
```

```
>>> str(32)
'32'
>>> str(3.14159)
'3.14159'
```

* NÚMEROS RANDÔMICOS

→ DETERMINISMO
- mesmo cálculo
- mesmo resultado

→ NÚMEROS PSEUDORANDÔMICOS
- não são verdadeiramente randômicos.
- gerados por computação determinística.
- impossível de distinguir dos randômicos.

→ FUNÇÃO RANDOM
→ gera número "float" entre ZERO e UM
→ inclui 0.0 mas não 1.0

```
import random
For i in range(10):
    x = random.random()
    print(x)
```

→ função "randint"

```
>>> random.randint(5,10)
5
>>> random.randint(5,10)
9
```

} retorna inteiros entre os parâmetros inclusive estes.

→ função "choice"

```
>>> t = [1,2,3]
>>> random.choice(t)
2
>>> random.choice(t)
3
```

} Selecionar um elemento randômico dentre os elementos de uma sequência

* FUNÇÕES MATEMÁTICAS

```
>>> import math
>>> print(math)
<module 'math' (built-in)>
```

→ principais funções matemáticas

```
>>> ratio = signal_power / noise_power
>>> decibels = 10 * math.log10(ratio)
>>> radians = 0.7
>>> height = math.sin(radians)
```

DOT NOTATION
- Nome do módulo seguido de "PONTO" e nome da função.

```
>>> degrees = 45
>>> radians = degrees / 360.0 * 2 * math.pi
>>> math.sin(radians)
>>> math.sqrt(2) / 2.0
```

CAPÍTULO – 04

* CRIANDO FUNÇÕES

→ DEFINIÇÃO DE FUNÇÃO

- Especificar o NOME da função e a sequência de comandos que devem ser executados quando a função é chamada.

```
def imprimir_mensagem():
    print('Bom dia a todos')
    print("Boa tarde a todos")
    print('Boa noite a todos')
```

- def: palavra reservada que define a função
- imprimir_mensagem(): CABEÇALHO
- FUNÇÃO SEM ARGUMENTOS
- CORPO
- IDENTAÇÃO DE 4 ESPAÇOS

→ NOMES DAS FUNÇÕES

- mesma regra das VARIÁVEIS
- letras
- números, menos no primeiro caracter
- não pode ser palavra reservada

OBSERVAÇÃO

→ Pode-se utilizar ASPAS SIMPLES ou DUPLAS

```
>>> print(imprimir_mensagem)
<function imprimir_mensagem at 0xb7e99>
>>> print(type(imprimir_mensagem))
<class 'function'>
```

FUNÇÕES PODEM CHAMAR OUTRAS FUNÇÕES

SINTAXE PARA A CHAMADA DE FUNÇÕES É A MESMA
- FUNÇÕES INTEGRADAS
- FUNÇÕES DEFINIDAS PELO USUÁRIO.

CAPÍTULO – 04

DEFINIÇÃO E USO

- A execução de uma função é feita da mesma forma que outros comandos são executados.
- O objeto "função" é criado.
- Comandos dentro da função só são executados quando a função é chamada.
- Obrigatoriamente as funções devem ser definidas antes de serem chamadas.

FLUXO DE EXECUÇÃO

- definição de funções não altera o fluxo de execução do programa.
- CHAMADA DE FUNÇÃO:
  - a chamada de função dentro de um programa faz com que fluxo de execução execute os comandos dentro da função, ao invés de ir para a próxima linha de execução do programa.

PARÂMETROS E ARGUMENTOS

```
def print_texto(nome)
    print(nome)
    print(nome)
```

```
>>> print_texto('Letra')
>>> print_texto(17)
>>> print_texto(math.pi)
>>> print_texto('Letra' * 4)
>>> print_texto(math.cos(math.pi))
```

CAPÍTULO-04

* FUNÇÕES FÉRTEIS E FUNÇÕES VAZIAS

(fruitful and void functions)

- FUNÇÕES FÉRTEIS
  - ⇒ Tem valor de retorno
- FUNÇÕES VAZIAS
  - ⇒ Sem valor de retorno

```
x = math.cos(radians)
golden = (math.sqrt(5)+1)/2
```

```
>>> math.sqrt(5)
```

```
>>> resultado = print_texto('R')
>>> print(resultado)
None
```

None → Nonetype

OBSERVAÇÃO: Nonetype

None != "None"

None → Nonetype "None" → string

* POR QUE UTILIZAR FUNÇÕES

- Organizar grupos de comandos, que tornam o programa fácil de ler, entender e tirar erros;
- Eliminação de código repetido;
- Dividir longos programas em partes fáceis de dar manutenção;
- Reuso de código, já devidamente testado e aprovado.

#