***TÓPICOS***

1. [**Função print( )**](#_Função_print(_))
   * 1. [**Formatações da função print( )**](#_Formatações_da_função)
     2. [**Exibições de dados na função print( )**](#_Exibição_de_dados)
     3. [**Argumentos end e sep**](#_Argumentos_end_e)
     4. [**Formatação numérica na função print( )**](#_Formatação_numérica_na)
2. [**Funções embutidas do Python**](#_Funções_embutidas_do_1)
   1. [**Funções diversas**](#_Funções_diversas)
   2. [**Funções de análise**](#_Funções_de_análise)
3. [**Métodos**](#_Modificação_de_objetos) **do Python**
   1. [**Métodos diversos**](#_Métodos_diversos)
   2. [**Funções de modificação**](#_Funções_de_modificação)
   3. [**Funções de divisão**](#_Funções_de_divisão)
   4. [**Funções de junção**](#_Funções_de_junção)
4. [**Particularidades das funções**](#_Particularidades_das_funções)
5. [**Atribuições de variáveis**](#_Atribuição_de_variáveis)
   * 1. [**Tipos primitivos de variáveis**](#_Tipos_primitivos_de)
6. [**Operadores aritméticos**](#_Operadores_aritméticos)
   * 1. [**Ordem de procedência**](#_Ordem_de_precedência)
7. [**Bibliotecas**](#_Módulos_e_bibliotecas) **e módulos**
   * 1. [**Biblioteca math**](#_Biblioteca_math)
     2. [**Biblioteca random**](#_Biblioteca_random)
     3. [**Biblioteca time**](#_Biblioteca_time)
     4. [**Biblioteca operator**](#_Biblioteca_operator)
8. [**Estruturas condicionais**](#_Estruturas_Condicionais)
   * 1. [**Estrutura condicional if/else**](#_Estrutura_condicional_if/else:)
9. [**Cores no terminal**](#_Cores_no_terminal)

**PYTHON 3 - MÓDULO 1**

# **Função print( )**

Para escrevermos uma mensagem na tela, utilizamos a função print( ). Ex.:

**print('Olá, mundo!')**

## **Formatações da função print( )**

Todos esses comandos têm que estar dentro de chaves { }. Ex.: {var:20}

**:10** - Inclui 10 espaços antes ou depois do valor da variável.  
**:>10** - Posiciona o conteúdo à direita.  
**:<10** - Posiciona o conteúdo à esquerda.  
**:^10** - Centraliza o conteúdo no meio dos espaços.  
**:x10** - Substitui os espaços por 'x'.  
**Obs.:** Podemos utilizar ambos os comandos acima em conjunto. Ex.: :xˆ10 centraliza o conteúdo em meio a dez ‘x’.

## **Exibição de dados na função print( )**

**print(frase[10])** - Exibe o índice 10 da frase.

**print(frase[0][2])** - Exibe o caractere 2 do índice 0.

**print(frase[10:15])** - Exibe do índice 10 até o 14.

**print(frase[10:15:2])** - Exibe do índice 10 até o 14, pulando de dois em dois.

**print(frase[:5])** - Exibe do início até o índice 4.

**print(frase[7:])** -Exibe do índice 7 até o final da frase.

**print(frase[4::3])** - Exibe do índice 3 até o final da frase, pulando de três em três.

Ao utilizarmos **índices negativos**, o início irá se dará pelo final de nossa string. Ex.: **print([-1])** irá exibir o último componente de nossa string. Tal característica também tem aplicação em diversos outros tipos de dados, como tuplas e listas, que veremos posteriormente.

## **Argumentos end e sep**

O argumento end determina o que iremos colocar ao final da string exibida  
pela função print. Em caso de não colocarmos nada nas aspas, a linha  
simplesmente não é pulada. Ex.: **print(f'Início da frase...', end = '')  
print('fim')**

O argumento sep determina o separador de cada elemento contido na string. O separador padrão é o espaço. Ex.:  
**print(f'Início da frase', ‘FInal da frase’, sep= '...')**No caso acima, os dois elementos serão separados por reticências.

## **Formatação numérica na função print()**

Para definirmos quantas casa decimais queremos, utilizamos: **{x}:.2f**

O número 2 indica o número de casas decimais e o f o tipo primitivo que queremos. Como queremos apenas duas casas decimais após a vírgula, queremos exibir duas casas decimais do tipo float (f).

Para adicionarmos dois textos em sequência, utilizamos o **símbolo + ou ,**

O símbolo + também serve para somarmos dois valores numéricos.

# **Atribuição de variáveis**

Para atribuirmos um valor em uma variável, utilizamos o **símbolo =**

**nome = 'Matheus'**

**idade = 21**

Para realizarmos a leitura de um valor de entrada em uma determinada variável, utilizamos o comando:

**nome = input('Qual o seu nome?')**

**­**

## **Tipos primitivos de variáveis**

1. **int: Números inteiros positivos, negativos e o zero.**
2. **float: Números reais.**
3. **boolean: Condicional (True ou False)**
4. **str: String (Converte qualquer valor em texto)**

**Obs.:** Podemos também realizar multiplicar variáveis do tipo string, fazendo-as serem repetidas. Ex.:   
**x = str(input('Digite algo: '))  
Digite algo: #  
print(x \* 10)   
  
O resultado exibido será: '##########'**

# **Funções embutidas do Python**

## **Funções diversas**

* **range(start, stop, step)** - Fornece uma sequência numérica através dos índices passados por parâmetro. O stop é obrigatório, já start e step são opcionais.
* **round(número, ‘número de casas decimais’)** - Realiza a formatação de um número real.

## **Funções de análise de objeto**

* **len( )** - Checa o número de caracteres do objeto.

Obs.: A contagem do método len( ) começa pelo índice 1.

* **.max( ) -** Retorna o maior valor presente no objeto, seja numérico ou alfabético.
* **.min( ) -** Ao contrário do .max( ), retorna o menor valor presente no objeto.
* **.sum( ) -** Caso o objeto seja numérico, retorna a soma de todos os valores presentes nele.

# **Métodos do Python**

## **Métodos diversos**

* **.count('a')** - Conta o número de letras ‘a’ minúsculas que possuem no objeto.

Obs.: A contagem do método len( ) começa pelo índice 0.

Podemos ainda selecionar o índice de início e final para a verificação. Ex.: **'var'.count('a', 3, 7)**

* O método **.count( )** também pode ser substituído pelo operador '**in**', podendo ser usado das seguintes formas: **'a' in frase** ou **frase in 'a'**,porém irá retornar um boolean, informando True ou False.
* **.find( )** - Irá informar o índice em que o caractere passado por parâmetro começa. Caso o caractere em questão não seja encontrado, a função irá retornar -1.
* **.rfind( )** - Começa a verificação pela direita, enquanto **.find( )** começa a verificação pela esquerda. Caso o caractere em questão não seja encontrado, a função irá retornar -1.

## **Métodos de modificação**

* **.replace(valor1, valor2)** - Substitui todo valor 1 encontrado pelo valor 2.
* **.upper( )** - Deixamos todos os caracteres da variável em letras maiúsculas. Podemos também escolher um caractere específico, passando-o por parâmetro.
* **.lower( )** - Realiza a mesma função do .upper( ), porém transformando as letras maiúsculas em minúsculas.
* **.capitalize( )** - Transforma todas as letras maiúsculas da variável em minúsculas com exceção do primeiro caractere.
* **.title( )** - Transforma toda a variável para letras minúsculas, com exceção da primeira letra após cada espaço.
* **.strip( )** - Elimina todos os espaços desnecessários da string. Com **.rstrip( )** eliminamos somente os espaços da direita, já com **.lstrip( )**, os da esquerda.
* **.insert(pos, valor)** - Insere, na posição passada por parâmetro, o valor especificado no segundo argumento.

## **Métodos de divisão**

**.split( )** - Transforma a string em uma lista, onde cada espaço contido na string será utilizado como separador. Ou seja, um espaço indica o começo de um novo elemento. Nos parâmetros do método split( ), podemos informar o separador a ser utilizado, e o número de elementos a serem criados na lista. Ex.:

**frase = ‘Uma frase qualquer.’**

**lista = frase.split(‘espaço’, 1)**

**print(lista)**

O comando acima exibirá a lista: **[‘uma’, ‘frase qualquer.’]**. O último espaço não foi considerado pois passamos por parâmetro que queríamos apenas dois elementos em nossa lista.

## **Métodos de junção**

**'x'.join(lista)** - Realiza a junção de uma cadeia de caracteres contidos em uma lista. 'x' é o que irá separar cada cadeia de caracteres. Caso queira que sejam separados por um espaço, basta colocar um espaço em branco no lugar.

Obs.: Após utilizar a função **.join( )**, a lista é convertida para uma string novamente.

## **Métodos de verificação**

* **.isupper****( )** - Verifica se a string está em maiúsculo.
* **.isalnum( )** - Verifica se a string está em Alfanumérico (Números ou/e letras).
* **.isalpha( )** - Verifica se a string é alfabética.
* **.isdigit( )** - Verifica se todos os caracteres na string são dígitos, incluindo dígitos do sistema decimal, superscripts e subscripts.
* **.isdecimal( )** - Verifica se todos os caracteres na string são dígitos do sistema decimal, excluindo caracteres que representam números em outras línguas e caracteres especiais como superscripts e subscripts.
* **.isnumeric( )** - Verifica se todos os caracteres na string são numéricos, incluindo caracteres que representam números em outras línguas, dígitos do sistema decimal, superscripts e subscripts.
* **.isidentifier( )** - Verifica se a string for um identificador válido no Python.
* **.islower( )** - Verifica se a string está em minúsculo.
* **.isprintable( )** - Verifica se todos os caracteres da string forem imprimíveis ou se a string estiver vazia.
* **.isspace( )** - Verifica se há apenas caracteres de espaço na string.
* **.istitle( )** - Verifica se a sequência for uma sequência com título.

Ex.: **x = str(input('Digite algo: '))   
print(x.isalnum())  
print(x.isalpha())  
print(x.isnumeric())  
print(x.isspace())**

## **Particularidades das funções**

Podemos também combinar mais de uma função, utilizando-a na mesma linha de comando. Ex.: **'var'.upper( ).count('O')**

Obs.: Strings são imutáveis, a não ser que utilizemos funções para modificá-las.

# **Operadores aritméticos**

1. **Adição**: +
2. **Subtração**: -
3. **Multiplicação**: \*
4. **Divisão**: /
5. **Potência**: \*\*
6. **Divisão inteira**: //
7. **Módulo** (Resto da divisão): %
8. **Igualdade**: ==
9. **Diferença**: !=

Obs.: O módulo de uma divisão com o dividendo maior que o divisor, é sempre igual ao dividendo. Ex.: 2 % 3 == 2

## **Ordem de precedência**

1. **()**
2. **\*\***
3. **\*, /, //, %**
4. **+, -**

**Obs.**: No caso de haver mais de uma operação da mesma ordem de precedência na expressão, sempre realizamos as operações da esquerda para a direita.

# **Bibliotecas e módulos**

Em Python, iremos trabalhar com pacotes contendo módulos em seus armazenamentos, de forma a não se ocupar espaço desnecessários na máquina.

Para importarmos um pacote a ser utilizado, será usado o comando:

**import ‘módulo’**

Para importar um módulo específico contido nesse pacote, utilize o comando:

**from ‘módulo’ import ‘x’**

Chamamos de biblioteca, qualquer projeto externo contido em um único módulo ou pacote. Há também as bibliotecas padrão, que já vem contidas na instalação do Python, sendo consideradas "parte da linguagem".

## **Biblioteca math**

Um exemplo de biblioteca padrão do Python é a biblioteca **math.** Nela estão contidas funções como:

* **.ceil() - Arredonda um valor para cima.**
* **.floor() - Arredonda um valor para baixo.**
* **.trunc() - Não arredonda o valor, eliminando tudo após a vírgula.**
* **.pow() - Calcula a potência de um número.**
* **.sqrt() - Calcula a raiz quadrada de um número.**
* **.factorial() - Calcula o fatorial de um número.**

Para importarmos uma função de tal biblioteca, iremos utilizar o comando:

**from math import sqrt**

## **Biblioteca random**

Exemplos de funções do módulo **random**:

**random.choice(lista)** - É sorteado aleatoriamente um dos elementos passados por parâmetro.

**random.choices(lista, [peso1, peso2, peso3], k=3)** - K é o tamanho da lista. Podemos também, definir o peso que cada um dos elementos terá ao ser sorteado. Nesse caso, os valores podem se repetir dentro da lista.

**random.shuffle(lista)** - Embaralha aleatoriamente os valores contidos em uma lista.

## **Biblioteca time**

**sleep( )** - Pausa o processamento do programa por um determinado número de segundo que será passado por parâmetro.

## **Biblioteca operator**

**itemgetter( )** - Chama o objeto que foi passado por parâmetro. Caso sejam passados múltiplos itens, retorna uma tupla contendo os mesmos. Quando lidamos com elementos de múltiplos objetos, ao passar um índice por parâmetro, estaremos chamando o objeto com esse índice de dentro do elemento. Ex.:

**lista = [(‘a’, ‘b’)]**

**itemgetter(0) - Chama a letra ‘a’**

**itemgetter(1) - Chama a letra ‘b’**

Junto desse módulo, muitas vezes iremos utilizar o argumento **key**. O argumento **key** é responsável por retornar as chaves de ordenação que serão utilizadas. Ex.:

**letras = ["c", "B", "d", "A"]**

**ordemAlfabetica = sorted(letras, key=str.lower( ))**

No exemplo acima, a função **.lower( )** é utilizada como argumento **key**, onde irá retornar uma lista passando os elementos para letra minúscula, que será utilizada na ordenação. O argumento **key**, além de funções, também pode passar objetos. Um exemplo muito comum, é com a utilização do método **itemgetter( )**.

# **Estruturas Condicionais**

## **Estrutura condicional if/else**

**if 'método':**

**bloco True**

**else:**

**bloco False**

Uma estrutura condicional simples possui somente uma condição (**if**) e quando tempos mais de uma (**if e else**), chamamos de condicional composta.

Obs.: Todo o comando que se encontrar indentado, fará parte da estrutura condicional.

# **Cores no terminal**

Para atribuirmos uma cor ou fonte a um certo texto no terminal, devemos utilizar o seguinte código: **‘barra inversa’033['estilo';'tipo';'fundo'm**

No código acima, ‘estilo’ define a cor do texto, ‘tipo’ a fonte e ‘fundo' a cor de fundo.

Para escolhermos o ponto em que tal customização se encerra, devemos colocar ao final o mesmo código: **‘barra inversa’033[m**

Se pode encontrar os códigos para o estilo, tipo e fundo na imagem a seguir:

