**Python Para Data Science**

**Curso 1 – Primeiros Passos**

1. **Aula 1 – Função e Linguagem de Alto Nível:**
   1. Interpolação de str com f’{}’: Podemos fazer interpolação/concatenação entre str e texto em uma única linha usando essa técnica:

nome = 'Bruno'  
idade = 21  
print(f'O nome é {nome} e a idade é {idade} anos')

* + 1. Lembrando que podemos fazer essa formatação dessa forma:

nome = 'Bruno'  
idade = 21  
print('O nome é {} e a idade é {} anos'.format(nome, idade))

* + 1. O resultado final no console é o mesmo.
  1. Input(): Recebe uma entrada do usuário e o programa só continua sua execução após sua entrada:

def saudacao():  
 nome = input('Qual o seu nome?\nR: ')  
 idade = int(input('E a sua idade, qual é?\nR: '))  
 return print(f'Olá, {nome}! Você tem a mesma idade que eu: {idade} anos')  
  
saudacao()



* + 1. Lembrando que quando criamos uma função, precisamos executar ela no final do código para que rode, por isso o saudação() lá no fim.
  1. Linguagens de alto nível são aquelas que parecem muito com a linguagem humana. Diferente daquelas que parecem mais com as de máquinas.
     1. A principal vantagem é a facilidade de entendimento.
     2. A desvantagem é que o interpreter precisa traduzir o que escrevemos para linguagem de máquina, para que o computador execute.
     3. No caso das linguagens de baixo nível, elas já estão em linguagem de máquina, não necessitando do interpreter e tendo um desempenho melhor.
  2. O que aprendemos:
     1. Aprendemos o que são variáveis;
     2. Criamos nossa primeira função;
     3. Entendemos a diferença entre linguagens de alto e baixo nível.

1. **Aula 2 – Parâmetros, Condicionais e Conversão de Tipos:**
   1. Função com parâmetro: Passamos como parâmetro para função algo que ela tenha que usar dentro do seu código para executar, mas esse parâmetro é definido fora dela:

def saudacao\_com\_parametros(nome\_da\_pessoa):  
 return print(f'Bom dia, {nome\_da\_pessoa}!')  
  
nome = 'Luiza'  
saudacao\_com\_parametros(nome)

* 1. Podemos usar condições dentro de funções:

idade = 20  
  
def verifica\_se\_pode\_dirigir(idade\_usuario):  
 if idade\_usuario >= 18:  
 return print(f'Você pode dirigir!')  
 else:  
 raise ValueError('Você não tem idade para dirigir!')  
  
verifica\_se\_pode\_dirigir(idade)

* + 1. Optei por colocar um erro, mas poderia colocar somente uma mensagem negativa.
  1. Podemos converter tipos de str para int, int para str, float para str e assim por diante.
     1. Inputs sempre recebem valores em str, mesmo que o valor passado seja um número, caso estamos esperando receber um número e utilizar esse valor em uma verificação que precise de números e não str, podemos fazer isso:
        1. Código recebendo str:

def verifica\_se\_pode\_dirigir\_sem\_parametros():  
 idade = input('Qual a sua idade?\nR: ')  
 if idade >= 18:  
 return print(f'Você tem permissão dirigir!')  
 else:  
 raise ValueError('Você não tem idade para dirigir!')  
  
verifica\_se\_pode\_dirigir\_sem\_parametros()

* + - 1. Convertendo para int, part1:

def verifica\_se\_pode\_dirigir\_sem\_parametros():  
 idade = input('Qual a sua idade?\nR: ')  
 idade = int(idade)  
 if idade >= 18:  
 return print(f'Você tem permissão dirigir!')  
 else:  
 raise ValueError('Você não tem idade para dirigir!')  
  
verifica\_se\_pode\_dirigir\_sem\_parametros()

* + - 1. Convertendo para int, part2 e como prefiro fazer:

def verifica\_se\_pode\_dirigir\_sem\_parametros():  
 idade = int(input('Qual a sua idade?\nR: '))  
 if idade >= 18:  
 return print(f'Você tem permissão dirigir!')  
 else:  
 raise ValueError('Você não tem idade para dirigir!')  
  
verifica\_se\_pode\_dirigir\_sem\_parametros()

* + 1. A mesma lógica se aplica para converter em str e float. Basta colocar str(variável)/float(variável).
  1. Tipos de operadores:



* + 1. Apostila de python e orientação a objetos da caelum: <https://www.caelum.com.br/apostila-python-orientacao-objetos/declarando-e-usando-variaveis/#entrada-do-usurio>.
  1. O que aprendemos:
     1. Criamos uma função que recebe um parâmetro;
     2. Aprendemos a usar as condicionais if e else;
     3. Convertemos a string (str) digitando através do teclado em um valor inteiro (int).

1. **Aula 3 – Listas, Repetições e Tipos Booleanos:**
   1. Quando vemos o tipo das listas recebemos: “list”
   2. A contagem dos elementos de uma lista começa no 0, portanto, se vc quiser pegar o primeiro elemento de uma lista, print o índice 0, o segundo é o índice 1 e assim por diante:

idades = [18,20,22,35,47]  
  
print(idades)  
print(idades[0])  
print(idades[1])



* + 1. Podemos pedir para o índice printar do índice x até o y, fatiando nossa lista, mas lembre-se que o fatiamento é exclusivo, portanto, se pedirmos para printar do 0 até o 2, ele irá printar somente o 18 e 20, que estão nas posições 0 e 1, excluindo o 2:

idades = [18,20,22,35,47]  
  
print(idades)  
print(idades[0])  
print(idades[1])  
print(idades[0:2])



* + 1. Dessa forma, se quisermos printar até o terceiro elemento da lista, precisamos falar para ele printar de 0 até 3:

idades = [18,20,22,35,47]  
  
print(idades)  
print(idades[0])  
print(idades[1])  
print(idades[0:2])  
print(idades[0:3])



* + 1. Se quiser printar de um determinado elemento até o final, podemos passar somente o elemento antes do “:”, ignorando o próximo:

idades = [18,20,22,35,47]  
  
print(idades)  
print(idades[0])  
print(idades[1])  
print(idades[2:])



* + 1. Se quisermos pegar o último elemento de uma lista, basta colocar -1 no índice:

idades = [18,20,22,35,47]  
  
print(idades)  
print(idades[-1])



* + 1. A ordem regressiva continua, ou seja, se quiser o penúltimo item, coloque índice -2, -3 para o anti-penúltimo e assim por diante.
  1. Podemos criar loops e laços dentro de funções, facilitando nosso processo.
     1. Ao invés de criar uma função que recebe uma idade e verifica se a pessoa pode ou não dirigir para cada uma delas em um for, printando isso:

def verifica\_se\_pode\_dirigir(idade\_usuario):  
 if idade\_usuario >= 18:  
 print(f'{idade\_usuario} anos de idade, TEM permissão dirigir!')  
 else:  
 print(f'{idade\_usuario} anos de idade, NÃO tem idade para dirigir!')  
  
idades = [18,20,12,22,15,35,47,17]  
  
for idade in idades:  
 verifica\_se\_pode\_dirigir(idade)



* + 1. Podemos criar uma função que recebe uma lista de idade e já com esse for dentro fazendo a verificação para cada uma:

def verifica\_se\_pode\_dirigir\_loop\_(idades):  
 for idade in idades:  
 if idade >= 18:  
 print(f'{idade} anos de idade, TEM permissão dirigir!')  
 else:  
 print(f'{idade} anos de idade, NÃO tem idade para dirigir!')  
  
idades = [18,20,12,22,15,35,47,17]  
verifica\_se\_pode\_dirigir\_loop\_(idades)



* + 1. Diminuindo o código e deixando mais simples e prático.
  1. Podemos criar listas com valores booleanos:

permissoes = []  
idades = [18,14,22,15,47]  
  
def verifica\_se\_pode\_dirigir(idades, permissoes):  
 for idade in idades:  
 if idade >= 18:  
 permissoes.append(True)  
 else:  
 permissoes.append(False)  
 print(permissoes)  
  
verifica\_se\_pode\_dirigir(idades, permissoes)



* + 1. Utilizamos o append() para adicionar os valores na lista.
  1. O que aprendemos:
     1. Criamos nossa primeira lista (link com operações com listas: <https://www.alura.com.br/artigos/listas-no-python>) no Python;
     2. Aprendemos a utilizar a estrutura de repetição for;
     3. Descobrimos o tipo booleano.