Funções

quinta-feira, 11 de agosto de 2022 16:29

Aula 52

Tudo que é usado no python é uma função.

Exemplo:
print("Hello World") -> função para escrever algo.
Variavel = **Input**("Digite um valor: ") -> Função para receber algum valor

No python conseguimos definir uma função para criarmos com a função:

def funcao() -> funcao é o nome da função que você deseja criar
    identação começa aqui.

Exemplo:

```
def funcao():
    print("Hello World")

funcao()
funcao()
funcao()
```

A função imprime 3 vezes o valor hello world

É usado quando estamos repetindo algo mais de uma vez, então precisamos usar isso em uma função.

quando criamos uma variavel dentro dos parâmetros da função é pedido quando usamos a função.

Exemplo:

def funcao(msg)
    print(msg)

funcao() -> a funcao vazia não retorna nada, então precisamos atribuir um valor.
funcao("Hello World") -> msg agora recebe Hello World

Na prática:

```
def funcao(msg):
    print(msg)

funcao()
funcao("Hello World")
```

O programa para de funcionar por conta do erro, por que não determinamos o valor.

quando determinamos mais de 1 variavel dentro da função precisamos também colocar dentro da função com valores.

def funcao(msg, nome)
    print(msg, nome)

funcao("Ola", "Luiz")

funcao("Alow", "Geronimo")

```
def funcao(msg, nome):
    print(msg, nome)

funcao("Olá", "Geronimo")
```

Outro metodo é usar os **parâmetros com valores padrão**:

Exemplo:

def funcao(msg = "Olá", nome = "Usuário")
    print(msg, nome)

funcao() -> dessa forma obtemos o resultado Olá Usuário por que não colocamos o valor dentro da função então vai usar o padrão.

funcao("Olá", "Geronimo")

```
def funcao(msg= "Olá", nome= "Usuário"):
    print(msg, nome)

funcao()
funcao("Olá", "Geronimo")
```

Existe também os parâmetros nomeados:

Exemplo:

def funcao(msg = "Olá", nome = "Usuário")
    print(msg, nome)

funcao(nome="Geronimo") -> Então a função pega o valor padrão Olá e usa o parâmetro nomeado Geronimo

```
def funcao(msg= "Olá", nome= "Usuário"):
    print(msg, nome)

funcao(nome = "Geronimo")
funcao("Olá", "Otavio")
```

```
def funcao(msg= "Olá", nome= "Usuário"):
    print(msg, nome)

funcao(nome = "Geronimo", msg="Hi")
funcao("Olá", "Otavio")
```

Dessa segunda forma invertemos os valores e o resultado é :

```
Hi Geronimo
Olá Otavio

Process finished with exit code 0
```

podemos então dentro da função colocar mais coisas para que ela faça:

Exemplo:

```python
def funcao(msg = "Ola", nome = "Geronimo"):
    nome = nome.replace("e", "A")
    msg = msg.replace("e", "A")
    print(nome,msg)

funcao()
```

Usamos o replace para mudar toda letra e dentro do nome e da mensagem para A, logo quando a funcao chamar os valores vao ficar assim :

```
GAronimo Ola

Process finished with exit code 0
```

Aula 53

Geralmente não usamos a função print dentro das funções.

quando usamos a função print dentro da função e jogamos a função dentro de uma variavel a variavel fica como se não tivesse valor.

Exemplo:

```python
def funcao(msg):
    print(msg)

variavel = funcao("Eu voltei")
print(variavel)
```

a variavel quando usar o print imprime o valor none como se a variavel não tivesse valor algum dentro dela.

```
Eu voltei
None

Process finished with exit code 0
```

Quando usamos o return dentro da função, o valor do parâmetro é passado para o return que quando jogado dentro de uma variável atribui esse valor a variavel, quando o return é encontrado dentro da função ela não desce mais depois do return.

Exemplo:

```python
def mensagem(msg):
    return msg

digitei = mensagem(20)
print(digitei)
```

o valor 20 foi atribuido ao parâmetro que o return passou para a variável digitei.

Fazendo um exemplo prático:

```python
def divisao(n1,n2):
    if n2 == 0 or n1 == 0:
        return

    return n1/n2

divide = divisao(8,2)

if divide:
    print(divide)
else:
    print("Divisão inválida.")
```

na divisão quando tem 0 o valor é none ou seja sem valor como vimos anteriormente logo podemos usar as estruturas condicionais para não ficar em none.

o return pode ter dentro do if e outro fora do if como se fosse um else

quando usamos a função com um return dentro sem parâmetro o que estiver no return será passado para a função do mesmo jeito.

Exemplo:

```python
def soma():
    return 2

var = soma()
print(var)
```

O valor de var agora é 2

```python
def soma():
    return 2

print(soma())
```

se usar o print na função acontece a mesma coisa.

existe uma possibilidade de retornarmos uma função dentro de outra função

Exemplo:

```python
def f(var):
    print(var)

def dumb():
    return f

var = dumb()
print(type(var))
```

var tem o tipo function, ou seja foi retornado o valor de função f para o var.

podemos retornar uma tupla dentro da função e chamar pelo indice:

Exemplo:

```
def dumb():
    return ("Luiz","Otavio")

var = dumb()
print(var[0],type(var))
```

Aula 54

Quando usamos funções de argumentos nomeados, após nomea-los os argumentos posteriores precisamos adicionar nomeados também, assim também funciona quando usamos a função.

Exemplo:

```
def rep(a1,a2,a3,a4,a5=10,a7):
    print(a1,a2,a3,a4,a5,a7)

rep(5,2,3,4,a4=6,a5)
```

ou seja os proximos argumentos precisam também ser nomeados por que caso contrário encontramos um erro.

quando usamos o return com mais de 1 argumento ele joga tudo isso em uma tupla

Exemplo:

```
def rep(a1,a2,a3,a4,a5=10):
    return a5,a1

var = rep(1,2,3,4,5)
print(var)
```

O valor de a5 = 5 e o valor de a1 = 1 logo o return joga para o var (5,1)

Em lista podemos adicionar os argumentos de uma lista dentro de outra lista com o empacotamento mas também podemos desempacotar com o mesmo metódo.

Exemplo:

```
lista= [1,2,3,5,4,6]
print(*lista)
```

dessa forma vemos a lista sem os colchetes

```
1 2 3 5 4 6

Process finished with exit code 0
```

Função *args no python

quando criamos uma função e queremos receber vários valores podemos usar o *args, porém essa função pega todos os valores que você adicionar como parâmetro e joga dentro de uma tupla

Exemplo:

```
def funcao(*args):
    print(args)

funcao(1,3,2,5,4)
```

Resultado:

```
(1, 3, 2, 5, 4)

Process finished with exit code 0
```

Se quiser mudar algum desses valores podemos usar um typecast para converter a tupla em uma lista

```
def funcao(*args):
    args = list(args)
    print(args)

funcao(1,3,2,5,4)
```

Resultado:

```
[1, 3, 2, 5, 4]

Process finished with exit code 0
```

e a partir disso podemos mudar algum valor da lista.

```
def funcao(*args):
    args = list(args)
    args[0]=20
    print(args)

funcao(1,3,2,5,4)
```

Quando usamos a função ***args** (nome mutavel) e usamos uma lista, essa lista vira o primeiro indice se não for mandada desempacotada.

Exemplo:

```python
def funcao(*args):
    print(args)

lista = [1,2,3,4,5]
funcao(lista,5)
```

então para resolver e esse problema e acessar a lista por indices normais usamos o *lista para desempacotar essa lista e jogar dentro de args.

Exemplo:

```python
def funcao(*args):
    print(args)

lista = [1,2,3,4,5]
funcao(*lista,5)
```

Podemos também dessa forma criar outra lista e usar como parâmetro desempacotada.

a função ****Kwargs** (nome que você escolher) são usamos para colocar argumentos nomeados dentro da função.

Os **kwargs servem para guardar valores de argumentos nomeados e o *args apenas numeros

os **kwargs salva os valores nomeados em um dicionário

Exemplo:

```python
def funcao(*args,**kwargs):
    print(args)
    print(kwargs)

lista = [1,2,3,4,6]
funcao(nome = "Neto",idade = 20, preço = 10)
```

Resultado :

```
()
{'nome': 'Neto', 'idade': 20, 'preço': 10}

Process finished with exit code 0
```

podemos então acessar os valores do dicionários usando os kwargs

Exemplo:

```python
def funcao(*args,**kwargs):
    print(args)
    print(f"Seu nome é {kwargs['nome']} e sua idade é {kwargs['idade']} anos ")

lista = [1,2,3,4,6]
funcao(nome = "Neto",idade = 20, preço = 10)
```

Então se quisermos saber se existe algum item dentro do dicionário podemos usar o get()

precisamos criar uma variável e usar essa função dentro da variável

```
def funcao(*args,**kwargs):
    print(args)

    nome = kwargs.get('nome')
    print(nome)

lista = [1,2,3,4,6]
funcao(nome = "Neto",idade = 20, preço = 10)
```

Se o valor não for encontrado dentro do dicionário então o valor vai ficar resultar em none

```
def funcao(*args,**kwargs):
    print(args)

    nome = kwargs.get('jumento')
    print(nome)

lista = [1,2,3,4,6]
funcao(nome = "Neto",idade = 20, preço = 10)
```

Resultado:

```
()
None

Process finished with exit code 0
```

Então podemos usar uma condicional para se o valor for none ele dar o resultado.

```
def funcao(*args,**kwargs):
    print(args)

    nome = kwargs.get('jumento')
    if nome is not None:
        print(nome)
    else:
        print("Esse valor não existe.")


lista = [1,2,3,4,6]
funcao(nome = "Neto",idade = 20, preço = 10)
```

Aula 55

Uma variável quando criada no python se for fora de uma função ela tem valor global ou seja ela pode ser usada dentro ou fora da função, porém quando usamos a mesma variável dentro da função com um valor diferente esse valor só funciona para dentro da função.

Exemplo:

```python
variavel = "nome novo"
def funcao():
    print(variavel)

def funcao2():
    variavel = "outro valor"


print(variavel)
```

logo o valor variavel dentro da func2 só tem função para dentro do escopo dela.
para deixar a variavel global que está dentro do func2 usamos global variavel e agora o valor da variavel para esse valor que está no escopo da função.

quando queremos mudar algo na variável podemos usar de outra forma de uma boa prática:

```python
variavel = "nome novo"
def funcao():
    print(variavel)

def funcao2(arg = None):
    arg = arg.replace("n","N")
    return arg

variavel_nova = funcao2(arg = variavel)
print(variavel_nova)
```

Aqui jogamos o valor da variável dentro da uma nova variável -> variavel_nova que recebe o parâmetro nomeado com a variável e ao receber esse valor tem um replace mudando a letra por outra.

Não podemos usar uma variavel global dentro de uma função e depois tentar mudar ela e usar.

Se quiser usar o valor de uma variável de uma função em outra devemos return ela para uma variável e usar na função2 como parâmetro

```python
"""
Crie uma função1 que recebe uma função2 como parâmetro e retorne o valor da função2 executada.

"""
def funcao1(arg):
    print(arg)

def funcao2(a):
    return a

valor = input("Digite seu nome: ")
variavel = funcao2(valor)
funcao1(variavel)
```

Aula 59

Quando precisamos chamar uma função dentro de outra função passamos ela como parâmetro e dentro da função usamos um nome qualquer para um

argumento função.

Exemplo:

```python
def mestre(funcao,*args,**kwargs):
    return funcao(*args,**kwargs)

def melancia(nome):
    return f"Seu nome é {nome} "

def funcao2(nome,saudacao):
    return f"Olá {nome} seja {saudacao}"

executando = mestre(melancia,"Geronimo")
executando2 = mestre(funcao2,"Geronimo","Bem-vindo!")
print(executando)
print(executando2)
```

Aula 60

Expressões lambdas são funções anônimas que são usadas para substituir algumas funções de forma mais fáceis.

Exemplo :
Em forma de função :

```python
def func(n1,n2):
    return n1+n2

var = func(2,2)
print(var)
```

Usando o Lambda:

```python
a = lambda x,y : x + y

print(a(2,2))
```

Podemos pegar uma lista e usar dentro de uma função para ordenar ela:

```python
lista = [
    ["p1",20],
    ["p2",30],
    ["p3",40],
    ["p4",50],
]
def func(item):
    return item[1]

lista.sort(key=func,reverse=True)
print(lista)
```

usando o lambda:

```
lista = [
    ["p1", 20],
    ["p2", 30],
    ["p3", 40],
    ["p4", 50],
]

lista.sort(key = lambda item: item[1])
print(lista)
```

```
lista = [
    ["p1", 20],
    ["p2", 30],
    ["p3", 40],
    ["p4", 50],
]

print(sorted(lista, key=lambda i: i[1], reverse=True))
```