Funções em Python

Na programação, funções são blocos de código que realizam determinadas tarefas que normalmente precisam ser executadas diversas vezes dentro de uma aplicação. Quando surge essa necessidade, para que várias instruções não precisem ser repetidas, elas são agrupadas em uma função, à qual é dado um nome e que poderá ser chamada/executada em diferentes partes do programa.

Declarações de função são feitas usando o comando def :

```
def oi():
    print("Olá Estou dando OI")

oi()
```

Funções em Python

Na programação, funções são blocos de código que realizam determinadas tarefas que normalmente precisam ser executadas diversas vezes dentro de uma aplicação. Quando surge essa necessidade, para que várias instruções não precisem ser repetidas, elas são agrupadas em uma função, à qual é dado um nome e que poderá ser chamada/executada em diferentes partes do programa.

Declarações de função são feitas usando o comando def :

```
def oi():
    print("Olá Estou dando OI")

oi()
```

Funções em Python

Podemos trabalhar com Funções com parametros:

```
def soma(a, b):
    return  a + b

c= soma(1,3)
print(c)

def identificacao(meu_nome,idade):
    print(f"Olá {meu_nome} Sua idade é: {idade}")

identificacao("João",34)
```

Funções em Python

Parâmetros Nomeados ou argumentos nomeados são onde definimos nomes e valores e sequencia de valores das nossas funções.

Vamos usar os mesmo exemplos das funções acima:

```
def identificacao(meu_nome,idade):
    print(f"Olá {meu_nome} Sua idade é: {idade}")

identificacao(idade=34, meu_nome="João")

salario = float(input("Digite seu Salário"))

def salario_descontado_imposto(salario, imposto=27.0):
    return  salario -   (salario    *   imposto *   0.01)

desconto = salario_descontado_imposto(salario, imposto=10.0)
print(desconto)
```

Funções em Python

Funções com Condicionais:

```
def validausuario(nome_usuario,senha):
    if nome_usuario == "admin" and senha == "python" :
        return "Usuário e senha Corretos Bem Vindos"
    elif nome_usuario == "admin":
        return "Usuário Correto"
    elif senha == "python":
        return "senha correta"
    else:
        return "Usuário e Senha Incorretos"

print(validausuario("",""))
```

Funções em Python

Podemos passar uma função dentro de outra função

```
def soma(a,b):
    return a + b

def calculo (num1,num2, fun=soma):
    return fun(num1,num2)

print(calculo(3,2))
```

Podemos adicionar mais 1 função ao calculo.

```
def subtracao(num1,num2):
    return num1 - num2

print(calculo(2,2,subtracao))
```

JCAVI
TREINAMENTOS EM TI

Funções em Python

Argumentos *args e **kwargs

Podemos passar um número arbitrário de parâmetros em uma função.
Utilizamos as chamadas variáveis mágicas do Python: ***args** e ****kwargs**.
Não é necessário utilizar exatamente estes nomes: *args e **kwargs. Apenas o asterisco(*), ou dois deles(**), serão necessários. Podemos optar, por exemplo, em escrever *var e **vars. Mas *args e **kwargs é uma convenção entre a comunidade que também seguiremos.

*args e **kwargs permitem passar um número variável de argumentos de uma função. O que a variável significa é que o programador ainda não sabe de antemão quantos argumentos serão passados para sua função, apenas que são muitos. Então, neste caso usamos a palavra chave *args.

Funções em Python

*args retorna vários elementos em uma tupla.

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

soma(1,4,5,6)

def soma_total(*args):
    total = 0
    for numero in args:
        total = numero + total
    return total

print(soma_total(4,4,2,25,234,2312))
```

Funções em Python

****Kwargs** também permite você passar um número indeterminado de parâmetros para uma função. Mas, diferente do ***args**, aqui os argumentos são passados com um identificador ou chave, similar a um dicionário Python. Isto nos facilita recuperar os argumentos. Veja no script a seguir:

```
def saudacoes(**kwargs):
    print(kwargs)

saudacoes(manha="bom dia",tarde="Boa Tarde",noite="Boa Noite"

def saudacoes_dia(**kwargs):
    for hora, saudacao in kwargs.items():
        print(f"Durante a {hora} dizemos {saudacao}")

saudacoes_dia(manha="bom dia",tarde="boa tarde")
```

Funções Decoradoras

Um decorador permite que você potencialize, modifique ou substitua completamente a logica de uma função ou método.

Vamos criar a função decoradora

```
def master(msg):
    def imprime():
        print("esse é a função principal")
        msg()
    return imprime
```

Segunda Função para utilizarmos juntamente com a função decoradora

```
def chama_funcao():
    print("Esta chamando a função Verdadeira")

chama_funcao()
```

Funções Decoradoras

Podemos decorar uma função de 2 maneiras:

1) Através de uma variavel

```
## Função decorada
chama_funcao = master(chama_funcao)
```

2) Através do decorador @

```
@master
def chama_funcao():
    print("Esta chamando a função Verdadeira")
```

Funções Decoradoras

Outros exemplos de funções decoradas:

```
def decoradora(valor):
        def imprime(*args):
            print("Soma Executada")
            return valor(*args)
        return imprime

@decoradora
def multiplica(a,b):
    return a * b

print(multiplica(2,3))
```

Funções Decoradoras

```
contador = 0

def contar_acessos(funcao_decorada):
    def nova_func(*args, **kw):
        global contador
        contador += 1
        return funcao_decorada(*args, **kw)
    return nova_func

@contar_acessos
def soma(a, b):
    return a + b

print(contador)
print(soma(2,2))
print(contador)
```

JCAVI
TREINAMENTOS EM TI

Exercícios Funções

Crie uma função que faça a soma de 3 números e imprima em tela o resultado da soma.

Crie um programa onde a função irá validar o nome e o CPF da pessoa estão corretos baseados nos dados apresentados. Valide com retornos em tela se o CPF está correto se o nome está correto.

Crie funções com argumentos *args e **kwargs e faça teste de saida dos argumentos e execute o iterador for nos elementos.

Crie 2 funções onde 1 será a função decoradora e a outra irá receber a função decorada. Teste pelo método decorado e pelo método através de variáveis.