Aula 08

LIP

Linguagem de Programação

Python

Funções

Um aspecto importante na resolução de um problema complexo é conseguir dividi-lo em subproblemas menores. Dessa forma, para resolver um determinado problema, uma tarefa importante é dividir o código em partes menores, fáceis de serem compreendidas e mantidas.

Já usamos diversas funções ou métodos sort(), len(), min(), max(), range...., diversos métodos nativos ou importados de bibliotecas, mas agora chegou a hora de fazermos nossas próprias funções ou métodos.

Python

Funções

Vamos evitar que os blocos do programa fiquem grandes demais e, por consequência, difíceis de ler e entender.

Sempre que possível permitir o reaproveitamento de códigos, implementados por você ou por outros programadores.

Evitar que um trecho de código seja repetido várias vezes dentro de um mesmo programa, evitando inconsistências e facilitando alterações.

Python

Escopo da função

```
def "nome da função"("parâmetros"):
    '''
    docstring contendo comentários sobre a função.
    Embora opcionais são fortemente recomendados. Os comentários
    devem descrever o papel dos parâmetros e o que a função faz.
    '''
    # corpo da função
    |
    | bloco de comandos
    |
```

Python

Esqueleto de programas funcionais

```
# função  principal
def main():
    '''
    Função principal, será a primeira a ser executado e
    será a responsável pela chamada de outras funções que
    por sua vez podem ou não chamar outras funçoes que
    por sua vez ...
    '''
    # corpo da função main
    |
    |  bloco de comandos
    |

# Declaração das funções
def f...
    '''
    docstring da função f
    '''
    # corpo da função f
    |
    |  bloco de comandos
    |

def g...
    '''
    docstring da função g
    '''
    # corpo da função g
    |
    |  bloco de comandos
    |

[...]

# início da execução do programa
main() # chamada da função main
```

Esqueleto de programas funcionais

Python

```
# função  principal
def main():
    '''
    Função principal, será a primeira a ser executado e
    será a responsável pela chamada de outras funções que
    por sua vez podem ou não chamar outras funçoes que
    por sua vez ...
    '''
    # corpo da função main
    |
    | bloco de comandos
    |

# Declaração das funções
def f...
    '''
    docstring da função f
    '''
    # corpo da função f
    |
    | bloco de comandos
    |

def g...
    '''
    docstring da função g
    '''
    # corpo da função g
    |
    | bloco de comandos
    |

[...]

# início da execução do programa
main() # chamada da função main
```

Apesar de não ser léxica e sintaticamente necessário para python ou programas interpretados, vamos seguir essa concepção por que outras linguagens funcionam dessa forma.

Funções sem retorno ou parâmetros

Python

```
def imprimirMsg():      #definição da função
    print("Olá Mundo!")
imprimirMsg()


def somar():
    res = int(input("Digite um numero: ")) + int(input("Digite outro numero: "))
    # A variável res é uma variável local, só é reconhecida dentro da função
    print(res)


somar()
print(res)
```

As variáveis criadas dentro da função, usadas como parâmetros e mesmo retornadas serão sempre locais, isso quer dizer que somente podem ser usadas dentro e durante a execução da função.

Funções sem retorno ou parâmetros

Python

```
#cuidado com o manuseio de variaveis globais em funções
a=1 #variável global
def somar():
    a=5
    print(a)
somar()
print(a)
```

```
5
1
```

Cuidado ao manusear variáveis globais dentro de funções, por ser um acesso de referência, após a execução o objeto volta a ter o resultado global

Funções sem retorno ou parâmetros

Python

```
inc=10
def incrementar():
    acc=0
    while inc<10:          # As variáveis globais não são reconhecidas diretamente
                           # dentro de funções
        print(acc, end=" ")
        acc+=1
incrementar()
print("")
print(inc)
```

```
0 1 2 3 4 5 6 7 8 9
10
```

Na medida do possível devemos evitar o uso de variáveis globais dentro de funções, que dificultam a compreensão, manutenção e reuso da função.

Se uma informação externa for necessária, ela deve ser fornecida como argumento ou parâmetros da função.

- Podemos definir argumentos que devem ser informados na chamada da função.

Funções sem retorno ou parâmetros

Python

```
inc=0
def incrementar():
    acc=0
    while inc<10 and acc<10: # As variáveis globais são reconhecidas diretamente
                             # dentro de funções, mas não é aconselhável atualiza-las
        print(acc, end=" ")
        acc+=1
    print()
    print(inc)
incrementar()
```

```
0 1 2 3 4 5 6 7 8 9

0
```

- Na medida do possível devemos evitar o uso de variáveis globais dentro de funções, que dificultam a compreensão, manutenção e reuso da função.
- Se uma informação externa for necessária, ela deve ser fornecida como argumento ou parâmetros da função.
- Podemos definir argumentos que devem ser informados na chamada da função.

Funções com parâmetros ou argumentos

Python

```
def somar(a,b): #argumentos ou parâmetros
    print(a+b)

somar(100,100)
somar(0.1,1.98)
somar("100","1010")
```

```
200
2.08
1001010
```

- Os parâmetros ou argumentos usados numa função são apenas locais e nesse caso também o python é que vai interpretar o tipo. Aqui é possível receber como parâmetros int, float e até str porque o operador “+” com strings concatena

Funções com parâmetros ou argumentos

Python

```
def somar(a,b): #argumentos ou parâmetros
    print(a+b)

somar(100,100)
somar(0.1,1.98)
somar("100","1010")
print(a,b)
```

```
Traceback (most recent call last):
  File "F:\projetosPython\aulas\funcoes.py", line 43, in <module>
    print(a,b)
          ^
NameError: name 'a' is not defined
200
2.08
1001010
```

- Se eu tentar usar os parâmetros fora da minha função somar(), o interpretador retornará um erro porque fora da função variáveis locais não existem

Funções com parâmetros ou argumentos

Python

```
def acrescentaItensLista(lista, item):
    lista.append(item)
    print(lista)

numeros = [10, 20, 30]
print(numeros)
acrescentaItensLista(numeros, 100)
print(numeros)
```

```
[10, 20, 30]
[10, 20, 30, 100]
[10, 20, 30, 100]
```

- Muito cuidado ao manipular estruturas mutáveis como listas e dicionários que acabamos de ver. Os parâmetros ou argumentos funcionam como atribuição direta e vai modificar tanto a variável fonte, quanto a variável que se quer modificar

Funções com parâmetros ou argumentos

Python

```
def acrescentaItensLista(lista, item):
    lista.append(item)
    print(lista)

numeros = [10, 20, 30]
print(numeros)
acrescentaItensLista(numeros.copy(), 100)
print(numeros)
```

```
[10, 20, 30]
[10, 20, 30, 100]
[10, 20, 30]
```

Para evitar a modificação das duas variáveis e somente aquela que deseja manipular utilize o método copy(), como já vimos, anteriormente

Retorno de funções

Python

```
def somar(a,b): #argumentos ou parâmetros
    print(a+b)

print(somar(4,4))
```

```
8
None
```

- A principio todas as funções criadas em python retornam alguma coisa, mesmo que seja um nenhum(None), agora se você precisa controlar o que e como isso será retornado, você deverá utilizar o comando return pra isso.

Retorno de funções

Python

```
def somar(a,b):
    res=a+b
    return res

print(somar(4,4))   8
```

```
def somar(a,b):
    res=a+b
    return res

print(somar(4,4))   8
print(type(somar)) <class 'function'>
```

Retorno de funções

Python

```
def somar(a,b):
    soma=a+b
    sub=a-b
    texto1="Resultado soma: "
    texto2="Resultado subtração: "
    return texto1, soma, texto2, sub

print(somar(4,4))
```

```
('Resultado soma: ', 8, 'Resultado subtração: ', 0)
```

É possível retornar diversos valores numa função, porém esse retorno se dá na forma de uma tupla, conforme o exemplo

Retorno de funções

Python

```
def somar(a,b):
    soma=a+b
    sub=a-b
    texto1="Resultado soma: "
    texto2="Resultado subtração: "
    return texto1, soma, texto2, sub

print(somar(4,4))
```

```
('Resultado soma: ', 8, 'Resultado subtração: ', 0)
```

É possível retornar diversos valores numa função, porém esse retorno se dá na forma de uma tupla, conforme o exemplo

Retorno de funções

Python

```
def somar(a,b):
    soma=a+b
    sub=a-b
    texto1="Resultado soma: "
    texto2="Resultado subtração: "
    return texto1, soma, texto2, sub


texto1,soma,texto2,sub=(somar(4,4))
print(type(somar(0,0)))
print(type(texto1))
print(type(texto2))
print(type(soma))
print(type(sub))
print(texto1,soma,texto2,sub)
```

Se não quiser esse tipo de variável de retorno ou precisar fazer uma formatação diferente dos dados retornados opte por modificar os dados para que se tornem globais

```
<class 'tuple'>
<class 'str'>
<class 'str'>
<class 'int'>
<class 'int'>
Resultado soma:  8 Resultado subtração:  0
```

Python

Funções Lambda

São usadas em funções simples criadas a partir da palavra reservada lambda, podem conter qualquer numero de parâmetros, mas retornam apenas um valor

```
soma = lambda a,b : a+b
print(soma(2,2))
```

```
4
```

Python

Funções Recursivas

Função recursiva é aquela que chama a si mesma. Nesse caso há de se tomar o cuidado de não permitir que sua execução seja infinita, estabelecendo condicionais para sua interrupção

```
def fatorial (n):
    if n==0 or n==1:
        return 1
    else:
        return n * fatorial(n-1) #chamada recursiva fatorial dentro dela mesma

numero=int(input("Digite um número para calcular o seu fatorial: "))
print(f"O fatorial de {numero} é {fatorial(numero)}")
```

```
Digite um número para calcular o seu fatorial: 5
O fatorial de 5 é 120
```