PYTHON

Dia 06A

Prof Flávio Sousa Silva
Recode Pro

Agenda

- Introdução
- Variáveis locais e globais
- Validação com funções
- Parâmetros opcionais e nomeação
- Funções como parâmetros
- Módulos

Introdução

- Funções são muito utilizadas em programação.
- Também podem ser chamadas de sub-rotinas.
- Seu benefício está em padronizar um determinado código
- A idéia é generalizar um código quando for possível
- Evitando assim, copiar sempre o mesmo código em várias partes do programa.

Introdução

- Em nossos códigos em python já utilizamos algumas funções prontas, como:
  - **print()**
  - **len()**
  - **input()**
  - **int()**
  - **float()**

Introdução

- Em python podemos criar nossas próprias funções.
- Funções possuem algumas característica:
  - Nome da Função: deve seguir as mesmas regras de nomeação das variáveis
  - Retorno da função: uma função pode ter retorno de sua operação.
    - Exemplo: Len()
  - **Mas uma função também pode não ter retorno nenhum.**

Introdução

- Funções também recebem valores como entrada para processamento.
- Chamados esses valores de parâmetros, que devem ser declarados em sua criação.
- Uma função pode ter qualquer tipo de paramêtro:
  - **Variáveis**
  - **Listas**
  - **Dicionários**
  - **Etc.**

Introdução

- Para criar uma função em python devemos utilizar a clausula **def**
- Ex:

```
def soma(a,b):
    print(a+b)
Invocando a função
soma(2,9)
soma(7,8)
soma(10,15)
```

- Atenção para a ordem de valores passados para a função

Introdução

- Existem situações em que uma função apresenta um retorno
- Para retorna um valor devemos utilizar a instrução **return** dentro do código da função
- **ex**

```
def soma(a,b):
    return(a + b)
 print(soma(2,9))
print(soma(7,8))
print(soma(10,15))
```

Introdução

- Ex: Função para verificar se um valor é par.

```
def espar(x):
    return(x%2==0)


print(espar(2))
print(espar(3))
print(espar(10))
```

Introdução

- Podemos utilizar funções dentro de outras funções.
- Um exemplo é usar a função **espar(x)** para saber se um valor é par ou ímpar.
- Criaremos a função chamada **par_ou_impar(x)** que usa a função espar(x).

Introdução

```
def espar(x):
    return(x%2==0)
def par_ou_impar(x):
    if espar(x)==True):
        return “par”
    else:
        return “ímpar”


>>>print(par_ou_impar(4))
>>>print(par_ou_impar(5))
>>>print(par_ou_impar(15))
```

Introdução

- Observe que utilizamos dois **return** no exemplo anterior.
- A instrução **return** fez com que a função pare de executar.
- Seu valor é retornado **imediatamente** para função que a chamou.
- Podemos entender a instrução **return** como uma **interrupção** da sua execução
- Assim com a instrução **break** dentro do **while** ou **for**

Introdução

- Função para retornar a pesquisa em uma lista.

```
>>>def pesquise(lista,valor):
>>>    for x,e in enumerate(lista)
>>>        if e == valor:
>>>            return x
>>>    return None
>>>L = [10,20,25,30]
>>>print(pesquise(L,25))
>>>print(pesquise(L,27))
```

Introdução

- Para ter boas funções tente fixar as seguintes regras:
  - Uma função deve resolver apenas um problema
  - E quanto mais genérica ela for, melhor será em longo prazo.
  - Tente defini-la sem utilizar a conjunção "e"
  - Se ela faz isso e aquilo já será um sinal ruim

Variáveis locais e globais

- Quando usamos funções, começamos a trabalhar com variáveis locais ou internas.
- Porém também podemos trabalhar com variáveis externas ou globais.
- A diferença está na visibilidade ou escopo dessas variáveis.
- Uma Variável local é definida dentro de uma função
  - **Existe apenas no escopo da função definida.**
  - **Não é vista fora dela.**

Variáveis locais e globais

- Uma Variável global é definida fora de uma função
  - **Pode ser vista por todas as funções do programa.**

```
>>>EMPRESA = “unidos LTDA”
>>>def imprime_cabecalho():
>>>    print(EMPRESA)
>>>    print(“-” * len(EMPRESA))
```

Variáveis locais e globais

- Variáveis globais devem ser utilizadas o **mínimo** possível em seus programas.
- Elas violam o **encapsulamento** de uma função.
- Uma variável **global** pode ser alterada por qualquer função
- Tornando a tarefa de saber **quem alterou** seu valor muito trabalhosa.
- Use para guardar valores constantes e que devam ser consultados por todas as funções
- Ou para variáveis de configuração de seu programa.

Validação com funções

- Funções são muito úteis para validar a entrada de dados.
- Um exemplo seria limitar o valor mínimo e máximo de valores inteiros recebidos
- Ao entrar com valores do valor mínimo ou máximo o programa pode solicitar uma nova entrada.

Validação com funções

```
def faixa_int(min,max):
    while True:
        v=int(input("digite valor"))
        if v<min or v>max:
            print("invalido - valor entre %d e %d"%(min,max))
        else:
            return print("Validado o valor %s"%v)
```

Parâmetros opcionais e nomeação

- Nem sempre precisamos e queremos passar todos os parâmetros definidos em uma função.
- Podemos declarar um valor padrão para caso o usuário não passa nenhum valor
- Porém, caso ele passe o valor como parâmetro, o valor padrão será alterado.

Parâmetros opcionais e nomeação

```
>>>def barra(n=40, caractere="*"):
>>>    print(caractere * n)
>>>barra(10) # faz com que n seja 10
>>>barra(10,"-") # faz 10 barras com
"-"
>>>barra( ) faz 40 barras com "*"
```

Parâmetros opcionais e nomeação

- Podemos combinar parâmetros opcionais com os obrigatórios.
- Porém ao usar esta combinação, eles não podem ser misturados entre si.
- Os parâmetros opcionais devem sempre ser os últimos da combinação.

Parâmetros opcionais e nomeação

- Python suporta a chamada de funções com vários parâmetros
- Porém até agora passamos **todos** os **parâmetros** das funções **seguindo** uma **ordem**
- Essa ordem é a mesma feita na **definição** da função.
- Caso não seguíssemos a mesmo ordem, teríamos problemas no processamento
- Ou seja, **a ordem importava.**

Função Lambda

```
def cubo1(num):
    resultado = num ** 3
    return resultado

def cubo2(num):
    return num ** 3

cubo3 = lambda num: num ** 3
```