Linguagem de Programação Python

Modularização

Professor: Ritomar Torquato

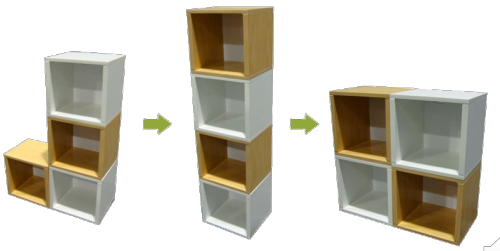
1

Modularização

Objetivos: Compreender a necessidade da criação de sub-rotinas para a organização do código fonte em módulos.

2

Modularização



3

Modularização

Cada **porção de conhecimento** deve possuir uma representação única.

4

Programas São Grandes

- Facilitar a construção de grandes programas, dividindo em **módulos**.
  - Programas são grandes!
- Também chamados de:
  - Subprograma;
  - Sub-rotina;
  - Função;
  - Procedimento; ou
  - Método

5

Modularização

- Em termos Gerais: Módulo ou Sub-Rotina

```
tipo nome_do_modulo_auxiliar([parâmetros]){

    [retorna valor]; //Opcional
}

tipo modulo_principal ([parâmetros]){
 nome_do_modulo_auxiliar([valores]);
}
```

6

Modularização

- Em termos de Python

```
def funcao_auxiliar ([parâmetros]):
    pass #Não faz nada


funcao_auxiliar([valores])
```

7

Modularização

```
def alo_mundo():
    print('Alô Mundo!')


alo_mundo()
```

Chama a função para execução.

8

Modularização

- Parâmetros: Entrada de dados para funções

```
def alo_nome(nome):
    print('Alô ' + nome + '!')


meu_nome = 'Maria'
alo_nome(meu_nome)
```

9

Modularização

- Parâmetros: Entrada de dados para funções

É possível definir um VALOR PADRÃO para parâmetros

```
def alo_nome(nome = 'Paulo'):
    print('Alô ' + nome + '!')


alo_nome()         # Imprime: Alô Paulo!
alo_nome("Maria") # Imprime: Alô Maria!
```

10

Funções

Sem Entrada de Dados

```
def diga_ola():
    print("Olá!")

diga_ola()
```

➡ Funções são chamadas para execução pelo seu nome e lista de argumentos necessários.

Com Entrada de Dados (Argumentos)

```
def diga_nome(nome, idade):
    print(f"Olá!{nome} tem {idade} anos")

diga_nome("João", 25)
```

11

Modularização

- Funções com retorno: Calcula e retorna;
  - Normalmente **não ler nada** do usuário e **não mostra nada** na tela;
  - Recebe os dados necessários por parâmetros;
  - Retorna usando `return`

12

Modularização

- Funções com retorno: Calcula e retorna;

```
def dobro(x):
    return x * 2

a = int(input("Digite um numero inteiro: "))
a = dobro(a)
print("O dobro é: ", a)
```

13

```
def percentual():
    v = float(input("Valor R$: "))
    p = float(input("Desejado %: "))

    v = p * (v / 100)

    print(f"Desejado R$: {v:.2f}")

percentual()
```

É um **ERRO** criar uma função que faz a TUDO (entrada, processamento e saída). Cada função deve realizar uma tarefa específica.

14

```
def percentual(valor, porcentagem):
    return valor * (porcentagem / 100)

v = float(input("Valor R$: "))
p = float(input("Desejado %: "))

v = percentual(v, p)

print(f"Desejado R$: {v:.2f}")
```

Usando FUNÇÕES específicas você melhora a leitura e fica mais fácil para reutilizar o código depois de feito.

15

Modularização

- Retornando vários valores

```
def dobro_e_triplo(n):
    dobro = n * 2
    triplo = n * 3
    return dobro, triplo

n = 4
x2, x3 = dobro_e_triplo(n)
print(f'{n} x 2 = {x2};')
print(f'{n} x 3 = {x3};')
```

É possível retornar vários valore em uma função separando por vírgula

16

Funções

Sem Retorno de Dados

```
def diga_ola():
    print("Olá, mundo!")

diga_ola()
```

Com Retorno de Dados

```
def dobro(n):
    return n * 2

d = dobro(27)
print(d)
```

Funções com retorno efetuam um cálculo e retornam o valor. NÃO devem ter comandos de **input()** ou **print()**.

17

Funções

Com Retorno de Dados

```
def minutos(horas, minutos):
    m = (horas * 60) + minutos
    return m

h = int(input("Horas: "))
m = int(input("Minutos: "))
m = minutos(h, m)
print(f'Minutos: {m}')
```

Os dados de entrada são recebidos como parâmetros (separados por vírgula) e a saída é feita com **return**.

18

Boa Prática em Python

Criar uma função **main()** que é a principal (onde o programa começa e termina.

```
def mensagem(msg):
    print(msg)

def main():
    mensagem("Alô, Mundo!")

if __name__ == '__main__':
    main()
```

19

Sobre o uso de FUNÇÃO

- Prefira sempre criar uma função para realizar tarefas específicas.
- No princípio, pode parecer um trabalho desnecessário mas acredite: a organização do código VALE A PENA.
- Não deve usar **input()** ou **print()** em funções que efetuam cálculos.
- Funções sem retorno podem usar o comando **print()**.

20

Modularização

Importante

Algumas exemplos trazem funções com prefixo **imprima_** ou **leia_** apenas para fins didáticos. O uso não é comum.

```
def leia_inteiro(msg):
    return int(input(msg))

def imprima_valor(msg, valor):
    print(f'{msg}{valor}')

a = leia_inteiro("Digite o valor para A: ")
imprima_valor("Valor de A: ", a)
```

21

Modularização

- A execução de um comando return encerra a função.

```
def dobro(x):
    return x * 2
    print(x)

print(dobro(4))
```

Este comando não será executado.

22

```
v = float(input("Valor R$: "))
p = float(input("Desejado %: "))

v = v * (p / 100)

print(f"Desejado R$: {v:.2f}")
```

Iniciantes em programação acreditam que sem funções o código fica menor. É um **ERRO** pensar assim.

23

```
def percentual(valor, porcentagem):
    return valor * (porcentagem / 100)

def main():
    v = float(input("Valor R$: "))
    p = float(input("Desejado %: "))

    v = percentual(v, p)

    print(f"Desejado R$: {v:.2f}")

if __name__ == '__main__':
    main()
```

Melhor ainda é seguir as boas práticas de programação em Python e criar sempre uma função **main()**

24

Variáveis Locais e Globais

- Escopo da variável: onde ela existe;
  - Local (função) ou Global (programa)
- Todas as variáveis em Python são local por padrão;
  - Para acessar uma variável global, usa-se a palavra reservada global
  - Não é recomendável uso de variáveis globais
- Parâmetros são variáveis locais à função.

25

Variáveis Locais e Globais

```
x = 42

def func():
    x = 1
    print ("Dentro da função:", x) # Imprime 1

func()
print ("Fora da função:", x) #Imprime 42
```

26

Variáveis Locais e Globais

```
x = 42

def func():
    global x
    x = 1
    print ("Dentro da função:", x) # Imprime 1

func()
print ("Fora da função:", x) #Imprime 1
```

27

Parâmetros ou argumentos

Os argumentos a, b, e c são variáveis locais para função imprime_3

```
def imprime_3(a, b, c):
    print(a, b, c)

x, y, z = 1, 2, 3
imprime_3(x, y, z)
```

Os valores de x, y e z são copiados, respectivamente, para a, b e c

28

Namespace

Coleção de nomes.

Namespaces são uma grande ideia — vamos ter mais dessas!

29

Teste de mesa

Quando o Namespace de uma variável for local, identifique usando [colchetes].

Namespace embutido: interpretador

Namespace global: módulo

Namespace local: funções

30

Teste de mesa

```
1  def mensagem(msg):
2      print(msg)
3
4  s = "Bem-vindo"
5  mensagem(s)
```

Memória		Tela
[mensagem]	s	(2) Bem-vindo
msg	(4) "Bem-vindo"	
(5) "Bem-vindo"		

31

```
1  def dobro(n):
2      resultado = n * 2
3      return resultado
4
5  valor = float(input('Valor: '))
6  d = dobro(valor)
7  print(f'O dobro de {valor} é: {d}')
```

Memória				Tela
[dobro]	[dobro]	valor	d	(5) Valor: 3.2 ↵
n	resultado	(5) 3.2	(6) 6.4	(7) O dobro de 3.2 é : 6.4
(6) 3.2	(2) 6.4			

32

```
1  def horas_e_minutos(minutos):
2      h = minutos // 60
3      m = minutos % 60
4      return h, m
5
6  min = int(input('Quantidade de minutos: '))
7  horas, minutos = horas_e_minutos(min)
8  print(f'{min} minutos é igual a {horas}h e {minutos}min')
```

Observe que a função retorna dois valores.

minutos existe nos Namespaces [global] e [horas_e_minutos], mas são **variáveis diferentes.**

Memória						Tela
[hm]	[hm]	[hm]	min	horas	minutos	(6) Quantidade de minutos: 190 ↵
minutos	h	m	(6) 190	(7) 3	(7) 10	(8) 190 minutos é igual a 3h e 10min
(7) 190	(2) 3	(3) 10				

Legenda:
hm – horas_e_minutos

← Se necessário, use legenda para identificar o Namaspace

33