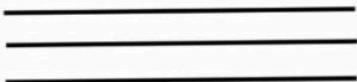


The background of the book cover is a painting of a black cat sitting in a pond. The cat is facing away from the viewer, looking towards a pink water lily flower on a green lily pad in the upper right corner. The water is depicted with dark, textured brushstrokes.

COMPOSITION BOOK

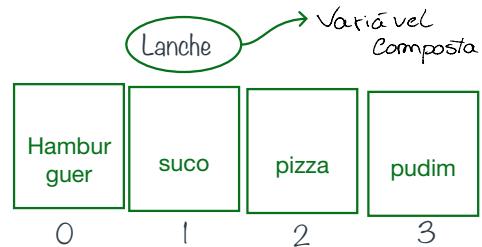


100 Sheets 200 Pages
9 ¾ x 7 ½ in / 24.7 x 19.0 cm

Variáveis Compostas

Tuplas ()

Tuplas são variáveis que 'guardam' mais que 1 espaço na memória



print (lanche [2])

Pizza

print (lanche [0:2])

Hamburguer, suco

print (lanche [:])

suco, pizza, pudim

print (lanche [-1])

Pudim

len (lanche)

4

for c in lanche:

print (c)

hamburguer

suco

Pizza

Pudim

O c é uma variável simples, por isso só aceita um "objeto" por vez

As Tuplas São Imutáveis

Variáveis Compostas

Listas []

Listas são mutáveis

`Lista.append('elemento')` → Com esse método podemos adicionar elementos na lista, no final da lista

`Lista.insert(0,elemento)` → Abre um espaço, o espaço 0, para colocar um novo elemento

`Del lista[3]` → Eliminar pela chave

`Lista.pop(3)` → Normalmente remove o último valor, mas podemos colocar a chave

`Lista.remove('elemento')` → Remove o elemento pelo 'nome'

`Valores = list(range(4,11))` → Crie uma lista de 4 a 10

`Valores.sort()` → Ordena os valores da lista

`Valores.sort(reverse=True)` → Coloca ao contrário

`Len(lista)` → Diz o tamanho da lista



Variáveis Compostas

Listas compostas []

pessoas = list()

pessoas.append(dados[:])

Comando para uma lista receber os elementos de outra lista (cópia)

Pessoas

| Dados | |
|--------|----|
| 'Nome' | 25 |
| 0 | 1 |

Pessoas

| | | |
|------------------|-------------------|-------------------|
| 'Nome' 25 0 1 | 'Nome2' 19 0 1 | 'Nome3' 32 0 1 |
| 0 | 1 | 2 |

pessoas = [['Nome', 25], ['Nome2', 19], ['Nome3', 32]]

Print(pessoas[0][0]) → Nome

Print(pessoas[1][1]) → 19

Print(pessoas[2][0]) → Nome3

Print(pessoas[1]) → ['Nome2', 19]

Variáveis Compostas

Dicionários {}

São estruturas de dados semelhantes às tuplas e listas, só que com elas conseguimos ter índices literais

| Dados | | |
|----------|-------|------|
| 'Silvia' | 25 | F |
| Nome | idade | sexo |

Dados = dict()

Dados = {'nome': 'Silvia', 'idade': 25} → declarando dicionários

Print(dados['nome']) → Silvia

Print(dados['idade']) → 25

Dados['sexo'] = 'F' → Para adicionar novos elementos nos dicionários, não utilizamos o "append" em vez disso, apenas colocarmos entre {} a nova variável e o valor que ela vai receber

Deldados['idade'] → Deleta o elemento e o que tem dentro

Print(dados.values()) → Retorna todos os valores (Silvia, 25, F)

Print(dados.keys()) → Retorna as chaves (nome, idade, sexo)

Print(dados.items()) → Retorna os dois (valores e chaves)

For k,v in dados.items() → Para cada chave e valor nos itens de dados

print(f'O {k} é {v}')
(O nome é Silvia)
(O idade é 25)
(O sexo é F)

Funções

Podemos criar funções personalizáveis

Def mostralinha():

```
print('-----')
```

função sem parâmetro

Esse ele não vai executar

Mostrarlinha()

Aqui, ele faz o print

Def mensagem(msg):

```
print('*30)
```

```
print(msg)
```

```
print('*30)
```

função com parâmetro

Esta função vai nos permitir adicionar texto dentro do print

Mensagem('sistema de alunos')

Quando chamarmos a função, ela vai executar tudo que colocarmos nela. Nesse caso, os 3 print

Mensagem é o nome da função, o que está em () é a msg que vai aparecer no print



Interactive Help (ajuda interativa)

Função help()

Podemos usar esta função diretamente no terminal, depois escolhemos qual função queremos um manual, ele nos da a descrição completa dessa função ou biblioteca. Para sair digitar 'quit'. Pode ser usada também no código, help(função).

Docstring

Podemos criar documentos para as nossas próprias funções.

Basta colocar 3 aspas duplas depois de def, por exemplo:

Def contador(i,f,p):

''' ''' '''

-> Faz uma contagem e mostra na tela.

: param i: inicio da contagem

: param f: fim da contagem

: param p: passo da contagem

: return: sem retorno

''' ''' '''

Resto do código

Parametros Opcionais

Se os parametros, **não receberem nenhum valor**, então eles **valem 0**. Exemplo:

Def somar(a = 0, b = 0, c = 0):

$$s = a + b + c$$

print(f'A soma vale (s)'

Caso ele **não** receba algum valor, o parâmetro **vazio** recebe 0, é **opcional** receber os 3 valores pedidos

Somar(3,2,5)

Assim essas 3 chamadas

Somar(8,4)

são válidas!

Somar()

Return

Def somar (a = 0, b = 0, c = 0)

$$s = a + b + c$$

return s

Nesse caso, colocarmos dentro de uma variável

Poderemos retornar para dentro de uma variável ou coloca dentro de um print: print(Somar(3,2,5))

R1 = somar(3,2,5)

R2 = somar(1,7)

Util para guardar vários resultados

Print(f'Meus calculos deram {r1} e {r2}')

Escopo de variáveis

Def teste(b):

a = 8

b += 4

c = 2

print(f"A dentro vale {a}")

print(f"B dentro vale {b}")

print(f"C dentro vale {c}")

Variáveis Locais

a 8

b 9

c 2

a = 5

Teste()

Print(f"A fora vale {a}")

escopo Local

Variável Global

a 5

escopo Global



Modulos

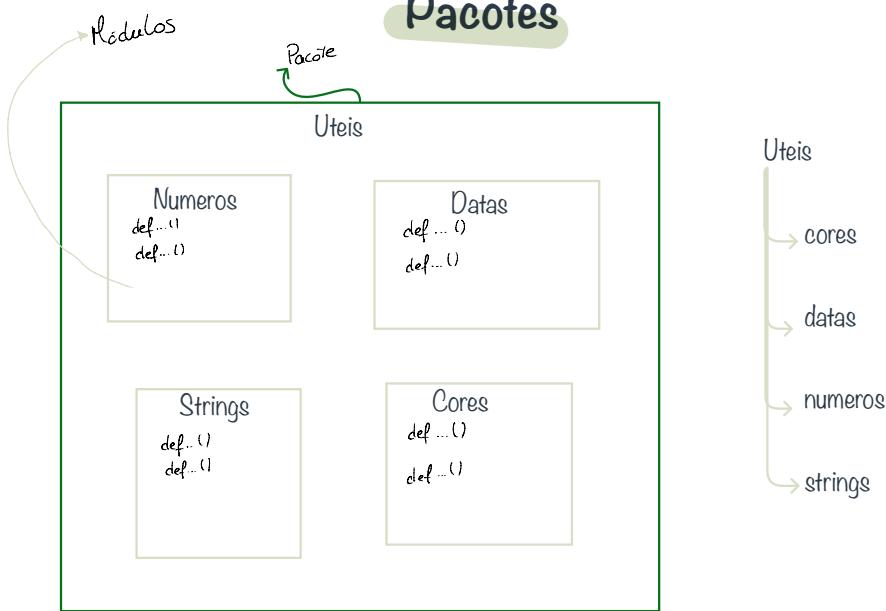
Modularização

- > Dividir um programa grande
- > Aumentar a legibilidade
- > Facilitar a manutenção

Vantagens

- > Organização do código
- > Facilidade na manutenção
- > Ocultação de código detalhado
- > Reutilização em outros projetos

Pacotes



Tratamento de erros e Exceções

Erro de sintaxe

`print(x)`
escrito errado

Exceção

`import uteis`

Não encontrado

ModuleNotFoundError



Exceção

`print(x)`
NameError

não existe

Exceção

`n = int(input('Num:'))`

O usuário digitou 'a' e não 8
ValueError

Exceção

`r = a / b`

Se b for 0

ZeroDivisionError

Exceção

`r = 2 / '2'`

TypeError

é um número?

Exceção

`lst = [3, 6, 4]`

`print(lst[3])`

Não tem index 3

IndexError