



Introdução a Python - Conceitos intermediários

Luiz Celso Gomes-Jr

Tipos de Variáveis

Variáveis possuem tipos e frequentemente precisamos fazer uma conversão entre estes tipos. O exemplo abaixo declara e converte várias variáveis.

```
area1 = 30
area2 = "45"
area3 = 35.5

print("Valores: ", area1, area2, area3)
print("Tipos: ", type(area1), type(area2), type(area3))

area1 = str(area1) #converte para String
area2 = float(area2) #converte para Float
area3 = int(area2) #converte para Inteiro

print("Novos valores: ", area1, area2, area3)
print("Novos tipos: ", type(area1), type(area2), type(area3))
```

Valores: 30 45 35.5

Tipos: <class 'int'> <class 'str'> <class 'float'>

Novos valores: 30 45.0 45

Novos tipos: <class 'str'> <class 'float'> <class 'int'>

Formatação de Strings

Frequentemente precisamos construir Strings que combinem diversos valores. Um método útil neste caso é o ***format***, que permite a substituição de pontos marcados com chaves ({}) por valores passados como parâmetro.

```
area1 = 45
area2 = 60

frase1 = "Área 1 = {}, Área 2 = {}".format(area1, area2)
print(frase1)
```

Área 1 = 45, Área 2 = 60.

Formatação de Strings

O método ***format*** também permite a formatação da aparência dos números. Abaixo construímos a *string* de forma a ter dois dígitos decimais para os valores de aluguel:

```
aluguel1 = 780
aluguel2 = 1100

print("Aluguel 1 = {:.2f}, Aluguel 2 = {:.2f}."
      .format(aluguel1, aluguel2))
```

Aluguel 1 = 780.00, Aluguel 2 = 1100.00.

Dicionários

Dicionário é um tipo de dado que nos permite dar nomes aos valores armazenados. Por exemplo, para armazenar as características de um imóvel, podemos definir um dicionário como no exemplo abaixo:

```
apartamento1 = {"endereco": "Av V. Guarapuava, 1000",  
                "area": 45, "aluguel": 800}
```

```
apartamento1
```

```
{'endereco': 'Av V. Guarapuava, 1000', 'area': 45, 'aluguel': 800}
```

Dicionários

Podemos preencher e acessar os valores do dicionário separadamente:

```
apartamento2 = {}  
  
apartamento2["endereco"] = "Av 7 de Setembro, 170"  
apartamento2["area"] = 53  
apartamento2["aluguel"] = 950  
  
print(apartamento2)  
print(apartamento2['aluguel'])
```

```
{'endereco': 'Av 7 de Setembro, 170', 'area': 53, 'aluguel': 950}  
950
```

Controle de fluxo - if

Ao escrever um programa, muitas vezes precisamos mudar a ação a ser tomada dependendo do estado das nossas variáveis. O operador mais utilizado neste caso é o ***if***. Por exemplo, abaixo imprimimos uma frase dependendo do valor do aluguel:

```
aluguel = 1300

if (aluguel < 900): print("Barato")
if (aluguel >= 900 and aluguel < 1400): print("Médio")
if (aluguel >= 1400): print("Caro")
```

Médio

Blocos de comando

Para executar vários comandos dentro de um mesmo *if* (ou outro comando), devemos escrever um **bloco de comandos**.

No Python, blocos de comando não possuem delimitadores.

Os blocos são definidos pelo alinhamento (indentação) dos comandos como no exemplo abaixo:

```
aluguel = 2300

if (aluguel > 2000):
    # início do bloco -- as linhas abaixo estão 4 espaços
    # à direita da linha do if
    print("Concedendo desconto..")
    desconto = 20 # 20% de desconto
    aluguel = aluguel - (aluguel * (desconto/100))

print("Valor final:", aluguel)
```

Concedendo desconto..

Valor final: 1840.0

Controle de fluxo - for

O comando **for** nos permite iterar sobre uma lista e executar ações associadas aos seus valores. No exemplo abaixo percorremos a lista de apartamentos definida anteriormente para calcular a média dos aluguéis:

```
total_aluguel = 0
numero_apartamentos = 0

for ap in apartamentos:
    total_aluguel = total_aluguel + ap["aluguel"]
    numero_apartamentos = numero_apartamentos + 1

media = total_aluguel/numero_apartamentos
print("Média: ", media)
```

Média: 875.0

Controle de fluxo - for

O comando **for** nos permite iterar sobre uma lista e executar ações associadas aos seus valores. No exemplo abaixo percorremos a lista de apartamentos definida anteriormente para calcular a média dos aluguéis:

```
total_aluguel = 0
numero_apartamentos = 0

for ap in apartamentos:
    total_aluguel = total_aluguel + ap["aluguel"]
    numero_apartamentos = numero_apartamentos + 1

media = total_aluguel/numero_apartamentos
print("Média: ", media)
```

Média: 875.0

List Comprehensions

List Comprehension é uma forma simplificada para se construir novas listas a partir de listas existentes. Por exemplo, abaixo definimos uma lista de números e criamos uma nova lista com estes números ao quadrado:

```
numeros = [5, 3, 7, 2]  
  
quadrados = [numero * numero for numero in numeros]  
  
print(quadrados)
```

```
[25, 9, 49, 4]
```

List Comprehensions

Usando a lista de apartamentos definida anteriormente, podemos criar uma lista com os preços por metro quadrado dos apartamentos:

```
print(apartamentos)

custo_por_m2 = [ap['aluguel']/ap['area'] for ap in apartamentos]

custo_por_m2
```

```
[{'endereco': 'Av V. Guarapuava, 1000', 'area': 45, 'aluguel': 800}, {'endereco': 'Av Sete de Setembro, 170', 'area': 53, 'aluguel': 950}]
```

```
[17.777777777777778, 17.92452830188679]
```

List Comprehensions

Os códigos abaixo são equivalentes:

```
custo_por_m2 = [ap['aluguel']/ap['area'] for ap in apartamentos]

custo_por_m2
```

```
[17.777777777777778, 17.92452830188679]
```

```
custo_por_m2 = []

for ap in apartamentos:
    custo = ap['aluguel']/ap['area']
    custo_por_m2.append(custo)
```

```
custo_por_m2
```

```
[17.777777777777778, 17.92452830188679]
```

Funções

Funções agrupam partes do código que têm um objetivo específico. As funções possuem parâmetros que permitem "personalizar" a execução a cada chamada. Abaixo definimos uma função que calcula o valor total a ser pago mensalmente em um apartamento:

```
def total_a_pagar(aluguel, condominio, taxa): # define 3 parâmetros
    total_taxa = condominio * taxa/100 # calcula valor da taxa
    total = aluguel + condominio + total_taxa
    return total # valor retornado pela função
```

```
# Chamadas à função:
```

```
total1 = total_a_pagar(1000, 300, 10)
```

```
total2 = total_a_pagar(1400, 450, 5)
```

```
print(total1, total2)
```

```
1330.0 1872.5
```

Classes

Classes são usadas para representar valores e operações de forma integrada. Abaixo definimos uma classe *Apartamento* para armazenar os valores de aluguel/condomínio e também oferecer uma função (método) para calcular o total.

```
class Apartamento:
    #definições de inicialização da classe
    def __init__(self, aluguel, condominio):
        # self neste caso é uma referência à instância da classe
        self.aluguel = aluguel
        self.condominio = condominio

    def calcular_total(self):
        return self.aluguel + self.condominio
# Inicializando uma instância
apl = Apartamento(1000, 300)

# Alterando o valor de um atributo
apl.condominio = 400

# Chamando um método
print("Total: ", apl.calcular_total())
```

Total: 1400

Exercícios!

- Revise o conteúdo e faça os exercícios do notebook:
04a-Outros Conceitos - Python.ipynb