Começando a Programar com o Python no QGIS

André Silva

GEOeduc

CAPÍTULO 2:

Construção de Funções Matemáticas

O Python é uma excelente linguagem para a construção de diversas aplicações de um modo geral, porém, é também, uma excelente linguagem para ser utilizada para fins matemáticos, como por exemplo, na geração de relatórios, gráficos, estatísticas e etc.

Na linguagem Python é possível utilizar diversas bibliotecas matemáticas para serem utilizadas, e atualmente diversas campos do conhecimento usam python para operações matemáticas. Python pode ser utilizada para, simples análises, geração de gráficos, inteligência artificial, análise de sentimentos, estatísticas para compra e venda de ações, neurociência entre outros.

Neste capítulos iremos abordar os seguintes assuntos:

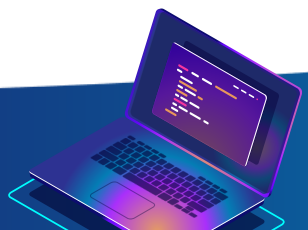
- Construção de funções matemáticas
- Diagrama de Classes

Operadores Matemáticos

Conforme já foi informado anteriormente o python pode ser utilizado como uma calculadora matemática avançada. Os operadores aritméticos em python, funcionam de um semelhante como conhecemos a matemática elementar. Abaixo segue um quadro que mostra as 4 principais funções matemáticas, a soma, subtração, multiplicação e divisão:

Operação	Operador
adição	+
subtração	-
multiplicação	*
divisão	/

Existem também outros operadores importantes, como exponenciação, parte inteira e módulo, conforme quadro abaixo:



Operação	Operador
exponenciação	**
parte inteira	//
módulo	%

Exemplos:

```
>>> #Soma
>>> 5+5
10
>>> #Subtração
>>> 150-148
2
>>> #Multiplicação
>>> 3*4
12
>>> #Divisão
>>> 63/7
9.0
>>> #Exponenciação
>>> 2**5
32
>>> #Extração da parte inteira da divisão
>>> 11//10
1
>>> #Resto da divisão
>>> 10%3
1
```

Valores Booleanos

Na linguagem Python, uma variável pode assumir valor chamado de booleano, ou seja *True* (verdadeiro) ou *False* (falso). Esses valores são úteis para representar o resultado de uma comparação. Veja:

```
>>> a = 3
>>> b = 4
>>> c = a < b # c recebe o valor da comparação a < b
>>> d = a > b # c recebe o valor da comparação a < b
>>> e = a == b # e recebe o valor da comparação a == b
>>> print("Valor de c: ", c)
Valor de c:  True
>>> print("Valor de d: ", d)
Valor de d:  False
>>> print("Valor de e: ", e)
Valor de e:  False
```

Operador **and**

Seja dois valores booleano A e B, o operador lógico *and* resulta em *True* apenas quando A e B foram ambos *True,* e retorna *False* caso contrário.

```
>>> A = True
>>> B = False
>>> print ("Valor de A and B: ", A and B)
Valor de A and B:  False
```

Operador **or**

Seja dois valores booleano A e B, o operador lógico *or* resulta em *False* apenas quando A e B foram ambos *False,* e retorna *True* caso contrário.

```
>>> A = True
>>> B = False
>>> print ("Valor de A or B: ", A or B)
Valor de A or B:  True
```

Operador **not**

O operador lógico *not* muda o valor de seu argumento, ou seja, *not true* é *False*, e *not False* é *True*.

Você sabia? A linguagem Python possui também operadores relacionais (de comparação) igualmente aos existentes da matemática.

- == (teste de igualdade): testa se duas coisas são iguais. Exemplo 10 == 10 o retorno será *True* e abacate == melancia o retorno será *False*.

Dica: Não confunda teste de igualdade (==) com atribuição em Python (=). Por exemplo em x = 20, estamos informando que a variável x, está recebendo o valor 20. Se fosse x == 20, o resultado seria *False,* x não é mesma coisa que 20.

- != (teste de diferença), a única função deste operador é testar a diferença entre dois valores. Exemplo: 10 != 20 retorna True, em outras palavras ele é o oposto do operador ==.
- >= (maior ou igual), exemplo: 10 >= 20 retorna *False.*
- <= (menor ou igual), exemplo: 10 <= 20 retorna *True*.
- > (maior), ex: 10 > 20 retorna *False*.
- < (menor), ex: 10 < 20 retorna *True*.

Funções Matemáticas

O Python possui um módulo de matemática que fornece as funções matemáticas mais comuns. Um módulo é um componente que contém uma coleção de funções relacionadas.

Antes de usar as funções no módulo, precisamos importá-las usando a instrução *import*:

Ao usar a instrução *import math*, um objeto de módulo é criado, chamado math(matemática). Ao exibir um objeto de módulo, informações sobre ele são exibidas:

```
>>> import math
>>> math
<module 'math' (built-in)>
```

Durante este tópico, iremos usar algumas métodos presentes no módulo *math,* para construir algumas funções matemáticas, como por exemplo o método: ceil, fabs, factorial, floor, exp, log, pow e sqrt.

ceil: Usado para arredondar um valor real para o inteiro mais próximo e acima.

```
>>> math.ceil(6.01)
7
>>> math.ceil(6.75)
7
>>> math.ceil(6.89)
7
```

fabs: Usado para remover o sinal de positivo ou negativo do valor, deixando apenas o próprio valor.

```
>>> math.fabs(-6.85)
6.85
>>> math.fabs(6.85)
6.85
>>> math.fabs(-788.8)
788.8
```

factorial: Usado para calcular o fatorial de um valor x.

```
>>> math.factorial(3)
6
>>> math.factorial(7)
5040
>>> math.factorial(5)
120
```

floor: Deixa somente aparte inteira de um valor, truncaremos o valor real para inteiro.

```
>>> math.floor(8.789546321)
8
>>> math.floor(189.7410)
189
>>> math.floor(753.1596)
753
```

pow: Usado para calcular a exponenciação de um valor.

```
>>> math.pow(3,3)
27.0
>>> math.pow(2,10)
1024.0
>>> math.pow(2,20)
1048576.0
```

EXERCÍCIO 1: OPERADORES MATEMÁTICOS

Essa atividade tem a finalidade de reforçar pontos abordados durante a aula que foi ministrada. Os pontos abordados, são de grande importância para desenvolvedores GIS, que irão criar seus próprios códigos dentro no QGIS diariamente.

1° Questão: Qual o resultado da função abaixo?

```
def main():
    a = True
    b = False
    print("Valor de 'a and b': ", a and b)

#-----
main()
```

2° Questão: Qual o resultado da função abaixo?

```
def main():
    a = True
    b = False
    print("Valor de 'a or b': ", a or b)

#-----
main()
```

3° Questão: Qual o resultado da função abaixo?

```
def main():
    a = 2
    b = 3

    print(a, b, b == a + 1)

#-----
main()
```

4° Questão: Qual o resultado da função abaixo?

```
def main():
    # modifique esse valor para ver o que o programa imprime
    # altura em metros e peso em Kg
    altura = 1.70
    peso   = 65.3

    # indice de massa corporal
    imc = peso / (altura * altura)

    print("O IMC = ", imc)
    print("Muito abaixo do peso  : ", imc < 17)
    print("Abaixo do peso normal : ", imc >= 17 and imc < 18.5)
    print("Peso dentro do normal : ", imc >= 18.5 and imc < 25.0)
    print("Acima do peso normal  : ", imc >= 25 and imc < 30)
    print("Muito acima do peso   : ", imc >= 30)
```

5° Questão: Qual informação (True ou False) completa o quadro abaixo?

or	A = True	A = False
B = True	True	
B = False	True	False

6° Questão: Qual informação (True ou False) completa o quadro abaixo?

and	A = True	A = False
B = True	True	False
B = False		False

Classes

Antes de entrarmos no PyQGIS propriamente dito, é importante entender certos conceitos relacionados às classes C ++ e Python. Qt e QGIS são escritos na linguagem C ++. A funcionalidade de cada widget Qt / QGIS é implementada como uma classe - com certas propriedades e funções. Quando usamos as classes PyQt ou PyQGIS, ele está executando o código nas classes C ++ por meio das ligações python.

Classes e Objetos: Uma classe pode ser pensada como um modelo. Você não pode usá-lo diretamente. Para usá-lo em um programa, você deve criar uma "instância" dele - que usa o modelo junto com os parâmetros fornecidos para criar uma instância da classe. Isso é conhecido como um objeto.

```
Obj = QMessageBox()
```

O QMessageBox é uma classe PyQt para criar uma caixa de diálogo com botões. Para usar a classe, você cria um objeto instanciando a classe. Aqui **Obj** é um objeto, que é uma instância da classe QMessageBox, criada usando os parâmetros padrão.

Inspeção de objetos: Para inspecionar objetos use o método **type()**, exemplo use: **type (Obj).** Caso queira verificar uma lista de atributos e métodos de um objeto, use **dir()**, exemplo: dir(Obj).

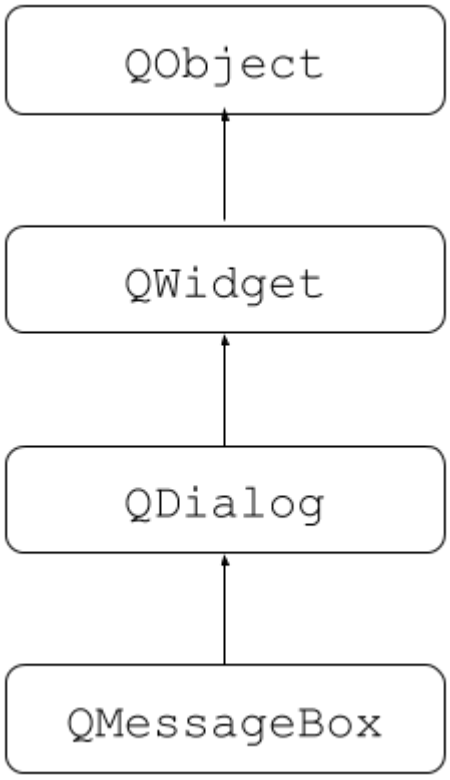
Métodos: As classes têm 2 tipos de atributos - atributos de classe e atributos de instância. O Qt fornece funções de acesso para atributos de instância. Para a classe QMessageBox, existe um método **text ()** para obter o atributo de texto da instância **Obj**.

```
Obj = QMessageBox()
Obj.setText('Click OK to confirm')
print(Obj.text())
```

Atributos de Classe: As classes também têm atributos de classe que são compartilhados em todas as instâncias. A classe QMessageBox possui os atributos Ok e Cancel, que podem ser consultados usando QMessageBox.Ok e QMessageBox.Cancel.

```
Obj = QMessageBox()
Obj.setText('Click OK to confirm')
Obj.setStandardButtons(QMessageBox.Ok | QMessageBox.Cancel)
```

Herança: QObject é a classe mais básica no Qt. Todos os widgets Qt e classes QGIS herdam do QObject. O widget mais básico é o QWidget. QWidget contém a maioria das propriedades usadas para descrever uma janela ou um widget, como posição e tamanho, cursor do mouse, dicas de ferramentas etc. A classe QDialog é a classe base das janelas de diálogo. QMessageBox é um QDialog especializado.



Por fim, experimente este exemplo simples que cria uma caixa de diálogo de confirmação usando o Pygis. Você pode digitar o código no Editor e clicar em Executar Script.

```
Obj = QMessageBox()
Obj.setText('Click OK para confirmar')
Obj.setStandardButtons(QMessageBox.Ok | QMessageBox.Cancel)
```

```
return_value = Obj.exec()
if return_value == QMessageBox.Ok:
   print('You pressed OK')
elif return_value == QMessageBox.Cancel:
   print('You pressed Cancel')
```

EXERCÍCIO 2: CLASSES (QgsProject)

Vamos aprendendo sobre novas classes do PyQGIS na medida que avançamos durante o curso. Como já vimos acima, classe em python é a definição de um tipo de objeto e de métodos (funções) associados a este objeto. Por exemplo um projeto, uma camada raster, uma camada vetorial, etc.

1º Vamos Criar e ler um projeto. Inicialmente vamos criar um projeto vazio, para isso iremos usar a classe QgsProject, fazendo o uso do método QgsProject.instance(). Chame seu projeto de primeiro_projeto.qgs, aonde também será necessário usar o método write(), pertencente a classe QgsProject.

```
>>> projeto= QgsProject.instance()
>>> projeto.write('C:/users/<seu usuário>/dados/primeiro_projeto.qgs')
>>> print(projeto.fileName())
```

2º Feche o QGIS e entre novamente para carregarmos o projeto que você criou anteriormente, porém agora usando Python.

```
>>> projeto=QgsProject.instance()
>>> projeto.read('C:/users/<seu usuário>/dados/primeiro_projeto.qgs')
```

Obs: Verifique se no canto superior esquerdo o projeto mudou para o projeto criado anteriormente.

EXERCÍCIO 3: CLASSES (QgsVetorLayer)

Vamos agora adicionar em nosso projeto 3 camadas já previamente criadas anteriormente usando os métodos de projeto addMapLayer e addMapLayers.

1º Caso tenha fechado, abra novamente seu projeto QGIS, usando python.

```
>>> projeto=QgsProject.instance()
>>> projeto.read('C:/users/<seu_usuário>/dados/primeiro_projeto.qgs')
```

2º Adicione as 3 camadas, criaremos um objeto de classe camada vetorial usando o método QgsVectorLayer() passando o caminho para o arquivo vetorial, o nome que nossa camada terá, e a biblioteca a ser usada (ogr nesse caso).

```
>>>relevo=QgsVectorLayer('C:/users/<seu_usuário>/dados/rel_elemento_fisiografico_natural_l.shp',"relevo", "ogr")
>>>extracao=QgsVectorLayer('C:/users/<seu_usuário>/dados/eco_ext_mineral_p.shp',”extracao”,”ogr”)
>>>limites=QgsVectorLayer('C:/users/<seu_usuário>/dados/lim_unidade_federacao_a.shp',"limites", "ogr")
>>> projeto.addMapLayers([relevo,extracao])
>>> projeto.addMapLayer(limites)
>>> projeto.write()
```

Obs: Observe que acima você poderia também ter escrito um número menor de linhas em python, inserindo os 3 layers de uma só vez: projeto.addMapLayers([relevo,extração,limites])

3º Por fim salve o projeto usando o método write().

```
>>> projeto.write()
```

4º (opcional) Podemos também adicionar dados do tipo raster, usando provedores do tipo TMS(TileMapService) ou WMS (WebMapService).

```
>>> uri="url=http://a.tile.openstreetmap.fr/hot/{z}/{x}/{y}.png&zmax=19&zmin=0&type=xyz"
>>> mts_layer=QgsRasterLayer(uri,'OSM','wms')
>>> projeto.addMapLayer(mts_layer)
>>> projeto.write()
```