UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
AUTOMATIZAÇÃO DE ANÁLISES URBANAS EM SIG
PYTHON PARA QGIS

Aula 2
2023-09-30

ANA LUISA MAFFINI
2023

Conteúdo - Aula 2

- Porque aprender programação?
- Apresentação do Python
- IDEs
- Indentação
- Comentários
- Variáveis
- Operadores binários
- Operadores aritméticos
- Manipulação de strings
- Condicionais
- Loops
- Funções
- Classes
- Importação
- Módulos e Bibliotecas

Porque estudar programação?

- Vivemos em um mundo cada vez mais imerso em dados.
- Dados geoespaciais são coletados em todo lugar.
- Pensar espacialmente inclui considerações como proximidade, adjacência, maneiras de medir o espaço geográfico, como elementos geográficos se relacionam uns com os outros, etc.
- Possibilidade de automatizar processos.
- Melhor compreensão de como os computadores e softwares são usados.
- Não repetir o mesmo trabalho manual mais uma vez.

Python

- Parte do sucesso do Python deve-se a facilidade de integração com códigos C, C++ e FORTRAN
- Python é uma linguagem interpretada
  - Uma instrução de cada vez
- Multiuso (Web, GUI, Scripting, etc.)
- Orientado a Objeto
  - Tudo é um objeto
- Maiúsculas e minúsculas

Python

- Criado em 1989 por Guido Van Rossum
- Python 1.0 lançado em 1994
- Python 2.0 lançado em 2000
- Python 3.0 lançado em 2008
- Python 2.18(?) é a última versão 2.x (obsoleta)
- Python 3.9 é a versão recomendada [2022] no QGIS atualmente

Zen do Python por Tim Peters

1) Bonito é melhor que feio.
2) **Explícito é melhor que implícito.**
3) **Simples é melhor que complexo.**
4) Complexo é melhor que complicado.
5) Linear é melhor do que aninhado.
6) **Esparso é melhor que denso.**
7) **Legibilidade conta.**
8) Casos especiais não são especiais o bastante para quebrar as regras.
9) Ainda que praticidade vença a pureza.
10) Erros nunca devem passar silenciosamente.
11) A menos que sejam explicitamente silenciados.
12) **Diante da ambiguidade, recuse a tentação de adivinhar.**
13) Deve haver um — e preferencialmente apenas um — modo óbvio para fazer algo.
14) Embora esse modo possa não ser óbvio a princípio a menos que você seja holandês.
15) Agora é melhor que nunca.
16) Apesar de que nunca normalmente é melhor do que *exatamente* agora
17) **Se a implementação é difícil de explicar, é uma má ideia**
18) Se a implementação é fácil de explicar, pode ser uma boa ideia
19) Namespaces são uma grande ideia — vamos ter mais dessas!

Python

Abra um interpretador Python interativo.
Inicie com o comando `python`.

Digite:

```
>>> print("Hello World")
Hello World
```

```
>>> a=5
>>> print(a)
5
```

Para sair do interpretador digite:
`exit()`

Ambientes de desenvolvimento integrado (IDE)

- PyDev
- PyCharm
- Spyder
- VS Code
- Jupyter Notebook
- WingIDE
- Komodo IDE

Indentação

O Python usa espaços em branco (tabulações ou espaços) para estruturar o código.

É recomendado utilizar quatro espaços como indentação padrão.

Python usa indentação para **legibilidade** E **funcionalidade**.

A maioria das linguagens não se importa com o recuo, mas a maioria dos humanos tende a agrupar coisas semelhantes.

```
if (foo)
    if (bar)
        baz (foo, bar);
else
    qux();
```

```
if (foo)
if (bar)
baz (foo, bar);
else
qux();
```

```
# Python code
if foo:
    if bar:
        baz (foo, bar)
    else:
        qux()
```

Comentários

Qualquer texto precedido por # (cerquilha) é ignorado pelo interpretador Python.

Geralmente é usado para a inclusão de comentários no código.

São importantes para a compreensão do código no futuro e por outros usuários.

Toda string que não for referida a uma variável é considerada um comentário.

```
# Isto é um comentário
```

```
“Isto também é”
```

```
”””
Aqui o comentário
possui mais de uma
linha.
”””
```

Variáveis

Para saber o tipo de uma variável pode se usar:
`type(x)`

String	str	armazena strings Unicode
Números	int	Números inteiros
Número	float	Números decimais
Null	None	O valor nulo do Python
Booleanos	bool	Valor booleano - True/False
Listas	list	Coleção ordenada []
Dicionários	dict	Associa valores a chaves e permite a rápida recuperação do valor {}
Tuplas	tuple	Semelhantes às listas mas não podem ser modificadas ()

Strings

```
# Isso é um string
name = "João da Silva"

# Isso também é um string
casa = 'Rua do Arvoredo, BR'

# Isso é um string de mais de uma linha
locais = '''Encontre João online
no facebook e instagram.'''

# Isso também é um string de mais de uma linha
info = """João é casado com Maria
e possui dois filhos e um cachorro."""
```

Números

```
# Números inteiros
ano = 2010
ano = int("2010")

# Número de ponto flutuante
saldo = 0.7958
saldo = float("0.7958")

# Número de ponto fixo
from decimal import Decimal
saldo = Decimal("0.79")
```

Booleanos

```
# Isso é um booleano
is_python = True

# Tudo no python pode ser convertido em booleano
is_python = bool("any object")

# Tudo isso é equivalente a Falso
these_are_false = False or 0 or "" or {} or [] or None

# Quase todo o resto equivale a True
these_are_true = True and 1 and "Text" and {'a':'b'} and ['c', 'd']
```

Listas

```
números = [1, 2, 3, 4, 5]

len(numbers)
# 5

numbers[0]
# 1

numbers[0:2]
# [1, 2]

numbers[2:]
# [3, 4, 5]
```

Dicionários

```
pessoa = {}

# Set by key / Get by key
pessoa['nome'] = 'João das Neves'

# Update
pessoa.update({
    'favoritos': [42, 'comida'],
    'gênero': 'masculino',
    })

# Qualquer objeto imutável pode ser uma chave de dicionário
pessoa[42] = 'número favorito'
pessoa[(44.47, -73.21)] = 'coordenadas'
```

Dicionários

```
pessoa = {'nome': 'João das Neves', 'gênero': 'masculino'}

pessoa['nome']
pessoa.get('nome', 'anônimo')
# Joäo das Neves Silva

pessoa.keys()
# ['nome', 'gênero']

pessoa.values()
# ['João das Neves', 'masculino']

pessoa.items()
# [['nome', 'João das Neves'], ['gênero', 'masculino]]
```

Operadores Binários

Operação	Descrição
a + b	Soma a e b
a - b	Subtrai b de a
a * b	Multiplica a por b
a / b	Divide a por b
a // b	Faz a divisão pelo piso de a por b, descartando qualquer resto fracionário
a ** b	Eleva a à potência de b
a & b	True se tanto a quanto b forem True, é a operação AND
a | b	True se a ou b for True, é a operação OR
a ^ b	Para booleanos, True se a ou b for True mas não ambos, é o EXCLUSIVE-OR
a == b	True se a for igual a b
a != b	True se a for diferente de b
a < b, a <= b	True se a for menor (menor ou igual) a b
a > b, a >= b	True se a for maior (maior ou igual) a b
a is b	True se a e b referenciam o mesmo objeto
a is not b	True se a e b referenciam objetos diferentes

Operadores Aritméticos

```
a = 10              # 10
a += 1              # 11
a -= 1              # 10


b = a + 1           # 11
c = a - 1           # 9


d = a * 2           # 20
e = a / 2           # 5
f = a ** 2          # 100
```

Manipulação de variáveis

```
a = [1, 2, 3]

b = a

c = list(a)


a is b
# True

a is not c
# True

a == c
# True
```

```
1 is 1 == True
# True

1 is not '1' == True
# True
```

Manipulação de strings

```
animais = "cães" + "gatos"
animais += "coelhos"
# cães gatos coelhos


frutas = ', '.join(['maçã', 'banana', 'laranja'])
# maçã, banana, laranja

data = '%s %d %d' % ('Set', 11, 2010)
# Set 11 2010

nome = '%(first)s %(last)s' %{
    'first': 'João',
    'last': 'Silva'}
# João Silva
```

Condicionais - if, elif, else

```
nota = 82
if nota >= 90:
    if nota == 100:
        print("A+")
    else:
        print("A")
elif nota >= 80:
    print("B")
elif nota >=70:
    print("C")
else:
    print("F")

# B
```

1) Abra o QGIS
2) Em complementos abra o Terminal Python
3) Clique em Mostrar editor
4) Escreva seu código ali
5) Ao terminar clique em Executar script

Loops - for

```
for valor in coleção:
 # Faz algo com o valor
```

```
for i in range(4):
    for j in range(4):
        if j>i:
            print((i,j))
```

```
for i in range(4):
    for j in range(4):
        if j>i:
            break
        print((i,j))
```

1) Abra o QGIS
2) Em complementos abra o Terminal Python
3) Clique em Mostrar editor
4) Escreva seu código ali
5) Ao terminar clique em Executar script

Loops - while

```
x = 0
while x < 100:
    print(x)
```

```
x = 0
while x < 100:
    print(x)
    x += 1
```

1) Abra o QGIS
2) Em complementos abra o Terminal Python
3) Clique em Mostrar editor
4) Escreva seu código ali
5) Ao terminar clique em Executar script

Função - range

A função **range** gera uma sequência de inteiros uniformemente espaçados.

O início, o fim e o intervalo entre os elementos podem ser fornecidos.

```
range(10)
range(0, 10)

list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

list(range(0,20,2))
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Funções

Cada função é uma regra que recebe zero ou mais entradas e retorna a saída correspondente.

Método de organização e reutilização de código.

No Python as funções são definidas usando **def**:

```
def my_function():
    """Documentação da função"""
    print "Hello World"

def add(x,y):
   return x + y
```

Classes

Encapsulam dados e suas respectivas funções.

Atributos atribuídos na declaração das classes devem ser imutáveis.

No Python as classes são definidas usando **class**:

```
class User(object):
    pass


class User(object):
    is_staff = False

    def __init__(self, name='Anônimo'):
    super(User, self).__init__()
```

Módulos e Importação

Normalmente grandes bibliotecas que consistem em um grande número de classes e funções.

A importação serve para permitir isolamento e reutilização de código

Ela adiciona referências a variáveis/classes/funções/etc. no namespace atual

É melhor importar apenas a função que você gosta do módulo, e não o módulo todo.

```
# Importa o módulo datetime no namespace atual
import datetime
datetime.date.today()
datetime.timedelts(days=1)

# Importa datetime e adiciona date e timedelta ao namespace atual
from datetime import date, timedelta
date.today()
timedelta(days=1)
```

Importação

```
# Renomeando Importações
from datetime import date
from my_module import date as my_date


# Evite fazer isso!
from datetime import *
```

Bibliotecas/Módulos Essenciais

- os, sys, shutil - manuseio geral do computador
- NumPy - faça o python ser como um “matlab”. Matrizes numpy são muito adequadas para rasters
- pandas - estruturas de dados e ferramentas de análise de dados fáceis de usar
- matplotlib - biblioteca de plotagem
- SciPy - faça o python ser como um “matlab”
- datetime - tratamento de data e hora
- PySAL - conjunto de métodos analíticos espaciais

Convenções de importação

```
import numpy as np

import pandas as pd

import statsmodels as sm

import matplotlib.pyplot as plt
```