

# PYTHON

---

Python è un linguaggio di programmazione potente e divertente, che trova utilizzo sia nello sviluppo di applicazioni web che di software desktop. E' possibile ritrovarlo come interfaccia di programmazione di librerie tipo GDAL/OGR, JTS e GEOS. Le caratteristiche principali sono:

- chiarezza e leggibilità della sintassi
  - è orientato alla programmazione ad oggetti ma anche alla programmazione funzionale
  - gestione degli errori in fase di esecuzione
  - tipi dato dinamici e di alto livello
  - codice modulare e riusabile
  - disponibilità sterminata di librerie immediatamente disponibili
- (<https://pypi.python.org/pypi>)

--

## VERSIONI DEL LINGUAGGIO

Python nel corso della sua evoluzione è arrivato ormai alla sesta release della versione 3 (3.6.8) e la versione 2 non sarà a breve più supportata.

Questo è il motivo principale per la migrazione a QGIS3.

- QGIS2 > Python 2.7
- QGIS3 > Python 3.5
- [informazioni sulla Migrazione](#)

--

## CONSOLE

---

consente di interagire con QGIS usando Python

L'interprete python usato da QGIS può essere usato anche da riga di comando. L'importante è che alcune "variabili d'ambiente" siano opportunamente configurate. La distribuzione di QGIS contiene una riga di comando opportunamente configurata per l'utilizzo delle librerie OSGEO utilizzate dal programma

---

## PRIMO CODICE

---

```
>>> print ("CIAO DA QGIS!") <invio>
```

print è una funzione i cui parametri sono contenuti dalle parentesi ()

Nel nostro caso il parametro è unico ed è costituito da una stringa di testo da stampare sulla console.

---

# I TIPI DI DATI

---

--

## Numeri

---

```
10      #integer (numero intero)
25.45   #float (numero decimale)

# Per trasformare una stringa in numero
# o troncare un float
int("10")
int(34.354)

# L'aritmetica ha una notazione convenzionale
10 + 1
10 * 5
10*(45/3)+8*17

# Divisione tra interi in python 2
5 / 2 #2

# se uno dei numeri è float il risultato è float
10 / 3.0

# questo è equivalente a sopra
10 / float(3)

# in python 3 il risultato è implicito
```

--

## Stringhe

---

```
"Ciao da QGIS!"

# Concatenare le stringhe
"Ciao" + " " + "da QGIS!"

# Formattazione di stringhe
"Ciao da %s" % "QGIS!"

# Il testo è racchiuso da virgolette singole o doppie
'Python è "divertente"'

# Stringhe multi-linea
print """Questa stringa
è suddivisa in
varie righe"""

# Caratteri speciale come a capo, la tabulazione o la barra inversa,
# possono essere inseriti per mezzo di una sequenza di escape che inizia
# con una barra inversa "\"

print "Questa stringa\nviene stampata\nin tre righe"
print "C:\\Users\\enrico"

# Si può usare una stringa grezza tralasciando le sequenze di escape
print r"C:\\Users\\enrico"

#Uso del set di caratteri esteso (unicode - UTF8) (Python 2)
print (u"questo è qgis")
```

--

## formattazione delle stringhe

---

il linguaggio dispone di [funzioni molto potenti](#) per assemblare tra loro le stringhe e formattare tipi non stringa (numeri, date)

```

a = 'Paolo'
b = "Verdi"
c = 172
d = 8.5676776

# concatenazione
print ("il mio cognome è " + b)
print ("mi chiamo "+a+" "+b+" alto "+str(c) " cm ed ho percorso "+str(d)+"
km")

# metodo "vecchio"
print ("il mio cognome è %s" % b)
print ("mi chiamo %s %s alto %d cm ed ho percorso %.1f km" % (a,b,c,d))

#metodo "nuovo"
print ("mi chiamo {0} {1} alto {:d} cm ed ho percorso {:.1f}
km".format(a,b,c,d))

```

--

## TIPO *Boolean*

---

```

True

False (None, [], {})

# operatori
not True # False
True and True # True
True and False # False
False and False # False
False or True # True
not (True or False) and True #False

1 == 1 operatore di uguaglianza
1 != 1 operatore di diversità
2 > 1 maggiore
2 >= 1 maggiore o uguale
1 < 2 minore
1 <= 2 minore o uguale

```

# VARIABILI

--

## Variabili

---

```
anno = "2017"

anno = 2017

print (anno)

anno_scorso = anno - 1

print (anno_scorso)
```

Le variabili in python hanno non sono "staticamente tipizzate", non devono essere dichiarate prima di usarle ne deve essere dichiarato il tipo. Sono dei contenitori che puntano a oggetti che potenzialmente possono essere di qualunque tipo. per conoscere il tipo dell'oggetto assegnato ad una variabile si usa type

```
type (anno)

<type 'int'>
```

---

## I TIPI *STRUTTURATI*

---

--

## Liste

---

una lista (list) e' un elenco ordinato di elementi

```

l = [3, 5, 4, 2, 1]
m = [[[0,3],[4,6],[5,7]],[[4,5],[6,8]]]

# Accesso agli elementi per posizione
# il primo elemento è 0, gli indici negativi partono dalla fine

l[0]          #3
l[0:3]        #[3, 5, 4]
l[-1]         #1
l[2:-2]       #[4, 2]
l[3:]         #[2, 1]
l[:-2]        #[3, 5, 4, 2]
m[0][2][0]    #5
m[1][1]       #[6,8]

```

la stessa notazione è valida per ricavare sottostringhe

```

"abcdefghijklmnopqrstuvwxyz"[-5:] # "vwxyz"

```

--

## Dizionari

---

un dizionario (dict) è un insieme strutturato di dati

```

d = {
    "chiave1": 10,
    "chiave2": 234.56,
    "chiave2": "contenuto",
    "chiave3": {
        "chiave3_1": "abcdef",
        "chiave3_2": "xyz"
    }
}

d["chiave2"]          # 234.56
d["chiave3"]["chiave3_2"] # "xyz"

```

--

## Manipolazione dei dati strutturati

---

```

10 = [] # lista vuota
11 = [1, 2]
12 = [3, 5, 6]

d0 = {} # dizionario vuoto
d1 = {"nome": "giuseppe", "anni": 50}

11 + 12          # [1, 2, 3, 5, 6]
11 + 4 + 13      # TypeError
11.append(10)    # [1, 2, 3, 5, 6, 10]
11.append(12)    # [1, 2, [3, 5, 6]]
3 in 12          # True
3 in 11          # False
len(11 + 12)     # 5

d1["cognome"] = 'verdi'
d1["anni"] = 60 # {"nome": "giuseppe", "anni": 60, "cognome": "verdi"}
d1.keys()       # ["cognome", "anni", "nome"]
d1.values()     # ["verdi", 60, "giuseppe"]
len(d1)         # 3

```

---

# Controllo del flusso

---

--

## IF THEN ELSE

---

```

latitude = 51.5
if latitude >= 0:
    zone_letter = 'N'
else:
    zone_letter = 'S'

print zone_letter

```

- l'indentazione è il sistema con cui Python raggruppa le istruzioni
- L'indentazione viene realizzata con spazi o tabulazioni
- ogni riga all'interno di un blocco base dev'essere indentata in ugual misura
- Usare un buon editor di testo per programmatori!

--

# Cicli

---

```
for carattere in "abcdefg":
    print (carattere)

for numero in [3, 5, 4, 2, 1]:
    print (numero)

testo = "abcdefg"
while testo:
    testo = testo[:-1]
    print (testo)
```

--

## Gestione delle eccezioni

---

```
try:
    print (100/0) #provoca ZeroDivisionError: integer division or modulo
by zero
except:
    print ("il codice contenuto in try provoca un errore")

try:
    print (100/0)
except Exception as e:
    print ("il codice contenuto in try provoca l'errore" + str(e))
```

## Funzioni, moduli e namespaces

---

--

## Funzioni esistenti (o predefinite)

---



```
# Funzioni esistenti (o predefinite)
print("stampa il numero",4)
sorted([3, 5, 4, 2, 1])
len(11 + 12)
type(a)

...
```

--

## Definizione della propria funzione

---

```
def ZonaUtm(longitudine, latitudine):
    numero_zona = ((longitudine + 180) / 6) % 60 + 1
    if latitude >= 0:
        lettera_zona = 'N'
    else:
        lettera_zona = 'S'
    return '%d%s' % (int(numero_zona), lettera_zona)
```

## Uso della funzione

```
ZonaUtm(11.55,45)    # 32
```

--

## Namespaces

---

```
def funzione1():
    var = 100
    print (var)

var = 50
print (var)
funzione1()
```

output:

```
50
100
```

verifica dei namespaces:

```
globals() #simboli accessibili globalmente
locals() #simboli accessibili localmente (all'interno di una funzione)
```

--

## test sui namespaces

---

```
global a
a = 0
test1()

def test1():

    def test2():
        c = 2
        print ('a',a)
        print ('b',b)
        print ('c',c)

    b = 1

    print ('b',b)
    test2()
```

--

## Moduli

---

```
# Importa un modulo con chiamata ad una funzione contenuta
import math
math.floor(10.6)

# Importa singoli elementi da un modulo
from os import path
path.join(r'C:\Users', 'Matt')
```

l'organizzazione modulare è una delle caratteristiche del linguaggio. I moduli possono essere:

- predefiniti, già compresi nella [dotazione di base del linguaggio](#)
- esterni, contenuti nei path di sistema di Python (PATH, PYTHONPATH). Possono essere preimportati o [importati da internet](#) tramite pip/setuptools
- definiti dall'utente in sottocartelle

--

## Uso dei moduli esterni

---

E' possibile usare moduli esterni configurando opportunamente la variabile d'ambiente PYTHONPATH in modo che essa punti ad una cartella contenente il codice del modulo

Oppure si può depositare manualmente la cartella contenente il modulo python dentro la cartella "site-packages"

A sua volta il modulo può dipendere da altri moduli python.

La comunità Python ha organizzato un repository dei moduli correntemente utilizzabili da python: <https://pypi.python.org/pypi> Il repository è utilizzabile tramite il comando [pip](#)

```
pip install [nome pacchetto]
```

Il comando provvede ad installare il pacchetto desiderato insieme alle sue dipendenze nella versione stabilita dai "*requirements*" del modulo, aiutando il programmatore a superare i conflitti tra versioni diverse dei pacchetti installati.

---

## Programmazione ad oggetti

---

--

## Classi ed istanze

---

- Classe
  - tipo di dato composto definito dall'utente in metodi ad attributi.
- Istanziare
  - creare un'oggetto di una determinata classe.
- Istanza
  - un'oggetto creato che appartiene ad una classe.
- Membri di una classe
  - Metodi

- funzioni che costituisce un comportamento dell'istanza
- Attributi
  - valori attribuiti ad un'istanza.
- Costruttore
  - metodo usato per definire nuovi oggetti.

--

## definizione di classe e membri

---

```
class rettangolo:

    l = 0
    h = 0

    def __init__(self, l, h):
        self.l = l
        self.h = h

    def area(self)
        return self.l*self.h

    def perimetro(self)
        return (self.l+self.h)*2

    def scala(self, scala)
        self.l = self.l*scala
        self.h = self.h*scala
```

--

## istanziare una classe

---

Una volta definita una classe è possibile .....

```
r = rettangolo(10,5) #istanziamento di un oggetto rettangolo
r.area()              #50
r.scala(2)
r.perimetro()         #60
r.l                   #20
r.h                   #10
```

--

# Ereditarietà

---

è la capacità di definire una nuova classe come versione modificata di una classe già esistente

```

class stanza:
    def __init__(self, lung, larg):
        self.lung = lung
        self.larg = larg
        self.nome = "stanza"

    def nome(self):
        return self.nome

    def area(self):
        return self.lung * self.larg

class cucina(stanza):
    def __init__(self, lung, larg):
        super().__init__(lung, larg)
        self.nome = "cucina"

class camera(stanza):
    def __init__(self, lung, larg, abitante):
        super().__init__(lung, larg)
        self.nome = "bagno"
        self.abitante = abitante

class edificio:
    def __init__(self, stanze):
        self.stanze = stanze

    def area_tot(self):
        area = 0
        for stanza in self.stanze:
            area += stanza.area()
        return area

    def abitanti(self):
        ab = 0
        for stanza in stanze:
            if hasattr(stanza, 'abitante'):
                ab += 1
        return ab

```

--

## ispezione degli oggetti

---

individuazione del tipo di oggetto

```
type (oggetto)
```

lista dei membri di un' oggetto

```
dir (oggetto)
```

restituzione di un attributo di un oggetto con il suo nome

```
getattr (oggetto, 'membro')
```

test se un attributo è presente nell'oggetto

```
hasattr (oggetto, 'membro')
```

---

# PyQGIS

---

--

## QgisInterface

---

QGIS è programmato in modo modulare in C++ ed espone classi e metodi di funzionamento in PyQGIS

```
# iface è l'oggetto principale di QGIS da cui derivano
# tutti gli altri oggetti che ne permettono il funzionamento
# è un "singleton" un oggetto con unica istanza
# della classe QgisInterface, istanziato come oggetto iface
iface.activeLayer().name()

# la maggior parte dei metodi richiamano altri oggetti
iface.activeLayer()

# quali metodi o attributi per un'oggetto:
dir(iface.activeLayer())
```

<https://qgis.org/api/classQgisInterface.html>

---

# Risorse

--

## Tutorial

---

- [Il tutorial ufficiale di Python](#)
- [Pensare da informatico. Versione Python](#)
- [Dive into Python - Python per programmatori esperti](#)
- [Google Python Course](#)

--

## Riferimenti

---

- [Cheatsheet - riferimento rapido python 2.7](#)
- [Cheatsheet - riferimento rapido python 3.5](#)
- [Python per esempi](#)

---

## Esercitazione 1

---

calcolo delle abbreviazioni del nome e del cognome del codice fiscale italiano:

- scomposizione di una stringa in vocali e consonanti
- uso delle consonanti in successione, e se non sufficienti uso delle vocali in successione

--

```
# -*- coding: utf-8 -*-
'''
CORSO DI GEOPROCESSING IN QGIS PER GEOLOGI
calcolo delle abbreviazioni del nome e cognome del codice fiscale
'''

import string

VOCALI = [ 'A', 'E', 'I', 'O', 'U' ]
CONSONANTI = list(set(list(string.ascii_uppercase)).difference(VOCALI))

def scomposizione(stringa):
    '''
    scomposizione nelle liste di consonanti e vocali che compongono la
    stringa in input
    '''
    stringa = stringa.upper().replace(' ', '')
```



```

consonanti = []
for car in stringa:
    if car in CONSONANTI:
        consonanti.append(car)

vocali = [ car for car in stringa if car in VOCALI ]
return consonanti, vocali

def abbreviazione(stringa):
    scomposizione_in_consonanti, scomposizione_in_vocali =
scomposizione(stringa)
    sequenza = scomposizione_in_consonanti
    while len(sequenza) < 3: #se la lunghezza è meno di 3 significa che
le consonanti non bastano e servono le vocali
        try: # pop toglie alla lista il primo elemento e lo restituisce
            sequenza.append(scomposizione_in_vocali.pop(0))
        except: # appende X per stringhe brevi
            sequenza.append('X')
    return ''.join(sequenza[:3]) # trasformazione della lista risultante
in stringa

nomi_da_abbreviare = ['Paolo', 'Riccardo', 'Alessia', 'Luisa', 'Ada',
'Liu', 'Bo']

for nome in nomi_da_abbreviare:
    print (nome, abbreviazione(nome))

```

[esercitazione1.py](#)

---

## Esercitazione 2 e 3

---

scrittura e lettura di un file di testo

--

### Scrittura

---

'''

CORSO DI GEOPROCESSING IN QGIS PER GEOLOGI

salvataggio di una stringa ad un file di testo

'''

```
import os

testo = '''
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Ut laoreet sem pellentesque ipsum rutrum consequat. Nunc iaculis tempor
aliquet.
Fusce imperdiet pharetra tellus, ut commodo lacus gravida et.
'''

destinazione = r"C:\Users\ferregutie\Downloads"

text_file = open(os.path.join(destinazione, 'mio_file.txt'), 'w')
text_file.write(testo)
text_file.close()
```

[esercitazione2.py](#)

--

## Lettura

---

```
'''
CORSO DI GEOPROCESSING IN QGIS PER GEOLOGI
lettura di una stringa da un file di testo
'''

import os
#from esercitazione2 import destinazione
destinazione = r"C:\Users\ferregutie\Downloads"

text_file = open(os.path.join(destinazione, 'mio_file.txt'), 'r')
testo = text_file.read()
text_file.close()

print (testo)
```

[esercitazione3.py](#)

## Esercitazione 4

---

stampare un'albero di files e directory. uso delle funzioni os:

- [os.path\(\)](#)
- [os.walk\(\)](#)
- [os.listdir\(\)](#)

--

```
'''
corso di geoprocessing in QGIS per geologi
procedura per stampare l'albero di files e directory
'''

import os

dir_sorgente = r"inserire un path"

print ("\nmetodo1")
for root, dirs, files in os.walk(dir_sorgente):
    for file in files:
        print(os.path.join(root,file))

print ("\nmetodo2")
def attraversa_dir(path, livello=0):
    file_e_dir = os.listdir(path)
    for elem in file_e_dir:
        spaziatura = ' '*livello*4
        if os.path.isdir(os.path.join(path,elem)):
            print (spaziatura + '['+elem+']')
            attraversa_dir(os.path.join(path,elem), livello=livello + 1)
            print ()
        else:
            print (spaziatura+elem)

attraversa_dir(dir_sorgente)
```

[esercitazione4.py](#)

## Esercitazione 5

calcolo della lunghezza di un segmento espresso come lista di punti con metodi diversi:

--

```
'''
CORSO DI GEOPROCESSING IN QGIS PER GEOLOGI
```

calcolo della lunghezza di un segmento espresso come lista di punti

```
'''
```

```
import math
```

```
def lunghezza(segmento):
```

```
    lunghezza_tot = 0
```

```
    for indice in range(0, len(segmento)):
```

```
        if indice == 0:
```

```
            pass
```

```
        else:
```

```
            x2 = segmento[indice][0]
```

```
            x1 = segmento[indice-1][0]
```

```
            y2 = segmento[indice][1]
```

```
            y1 = segmento[indice-1][1]
```

```
            lunghezza_tot += math.sqrt((x2-x1)**2 + (y2-y1)**2)
```

```
    return lunghezza_tot
```

```
def lunghezza_elegante(segmento):
```

```
    lunghezza_tot = 0
```

```
    for indice, p2 in enumerate(segmento):
```

```
        if indice == 0:
```

```
            pass
```

```
        else:
```

```
            lunghezza_tot += math.hypot(p2[0]-p1[0], p2[1]-p1[1])
```

```
            p1 = p2
```

```
    return lunghezza_tot
```

```
def lunghezza_pitonica(segmento):
```

```
    diffpt = lambda p: (p[0][0]-p[1][0], p[0][1]-p[1][1])
```

```
    lista_diff = map(diffpt, zip(segmento[:-1], segmento[1:]))
```

```
    lunghezza_tot = sum(math.hypot(*d) for d in lista_diff)
```

```
    return lunghezza_tot
```

```
polilinea = [[1.3,3.6],[4.5,6.7],[5.7,6.1],[2.9,0.6],[3.4,2.1],[9.5,2.7]]
```

```
print ("lunghezza",lunghezza(polilinea))
```

```
print ("lunghezza_elegante",lunghezza_elegante(polilinea))
```

```
print ("lunghezza_pitonica",lunghezza_pitonica(polilinea))
```

```
for nome_funzione in ['lunghezza', "lunghezza_elegante",
```

```
"lunghezza_pitonica"]:
```

```
    print (nome_funzione, globals()[nome_funzione](polilinea))
```

