# Sintesi del Laboratorio di Programmazione Python

Università di Trieste

## Indice

# 1 Introduzione a Python

Python è un linguaggio di programmazione interpretato ad alto livello, creato nel 1991. È particolarmente apprezzato per la sua semplicità, leggibilità e per il vasto ecosistema di librerie, specialmente nell'ambito della Data Science. Python supporta diversi paradigmi di programmazione, tra cui la programmazione orientata agli oggetti e la programmazione funzionale.

## 1.1 Hello World

Il classico esempio "Hello World" in Python:

```
print("Ciao, Mondo!")
```

## 1.2 La funzione print

```
Stampa semplice
print('Ciao, Mondo!')
Stampa con formattazione
x = 25
print('Ho {} anni.'.format(x))  # Stampa 'Ho 25 anni.'
```

## 1.3 Formattazione delle stringhe con .format()

Le doppie graffe  rappresentano l'ancora. Può essere comodo associare una chiave all'ancora scrivendo al suo interno una chiave univoca:

```
print('Valore 1: {A}, valore 2: {B}'.format(B=36, A=5))
```

# 2 Tipi di Dati

Python è un linguaggio non tipizzato, il che significa che non è necessario dichiarare esplicitamente il tipo di una variabile. Il tipo viene assegnato automaticamente in base al valore.

```
mia_var = 1        # intero (int)
mia_var = 1.1      # floating point (float)
mia_var = 'ciao'   # stringa (str)
mia_var = True     # booleano (bool)
mia_var = None     # valore nullo (None)
```

Il tipo di una variabile può essere verificato con la funzione `type()`:

```
type(mia_var) is int  # Verifica se mia_var    di tipo
   intero
```

### 2.1 Accesso alle stringhe per posizione

Si può accedere all'elemento i-esimo di una stringa così:

```
mia_stringa[2]   # Terzo carattere della stringa
mia_stringa[-1]  # Ultimo carattere della stringa
```

### 2.2 Slicing delle stringhe

Si può estrarre una sottostringa ("slice") di una stringa così:

```
mia_stringa[0:50]   # Dal 1   al 50   carattere
mia_stringa[30:50]  # Dal 30   al 50   carattere
mia_stringa[0:-1]   # Dal 1   al penultimo carattere
mia_stringa[1:3:2]  # Dal 2   al 4   carattere con
    intervallo di 2
mia_stringa[-1::-1] # Dall'ultimo carattere al primo con
    intervallo di -1
```

## 3 Operatori

### 3.1 Operatori di confronto

```
x == y  # Uguale
x != y  # Diverso
x > y   # Maggiore
x < y   # Minore
x >= y  # Maggiore o uguale
x <= y  # Minore o uguale
```

### 3.2 Operatori aritmetici

```
x + y   # Addizione
x - y   # Sottrazione
x * y   # Moltiplicazione
x / y   # Divisione
x % y   # Modulo (resto della divisione)
x ** y  # Esponenziale
```

### 3.3 Operatori logici

```
x < 5 and x < 10  # True se entrambe le condizioni sono
    vere
x < 5 or x < 4    # True se almeno una condizione    vera
not(x < 5)        # Inverte il risultato
```

# 4 Istruzioni Condizionali e Cicli

## 4.1 If-Else

In Python, l'indentazione è usata per definire i blocchi di codice. Ogni livello di indentazione è tipicamente di 4 spazi.

```
if (my_var > your_var):
print("My var is bigger than yours")
if (my_var-your_var) <= 1:
print("...but not so much")
elif (my_var-your_var) <= 5:
print("...quite a bit")
else:
print("...a lot")
```

## 4.2 Cicli

### 4.2.1 For

```
for item in mylist:
print(item)  # Itera su ogni elemento della lista
for i in range(10):
print(i)     # Stampa numeri da 0 a 9
for i, item in enumerate(mylist):
print("Posizione {}: {}".format(i, item))
```

### 4.2.2 While

```
i = 0
while i < 10:
print(i)
i = i + 1
```

# 5 Funzioni

Le funzioni in Python si definiscono con la parola chiave `def`.

```
def mia_funzione(argomento1, argomento2):
print("Argomenti: {} e {}".format(argomento1, argomento2)
    )
mia_funzione('Pippo', 'Pluto')  # Stampa: "Argomenti:
    Pippo e Pluto"
```

```
def eleva_al_quadrato(numero):
return numero * numero
risultato = eleva_al_quadrato(4)  # risultato = 16
print('Risultato: {}'.format(risultato))
```

### 5.1 Funzioni con valori di default per gli argomenti

```
def eleva_alla_n(numero, n=2):
return numero**n
```

### 5.2 Docstring delle funzioni

Una docstring è una stringa posizionata immediatamente sotto la definizione della funzione, racchiusa tra triple virgolette (""" ... """). Viene usata per documentare la funzione.

```
def somma(a, b):
"""
Calcola la somma di due numeri.
Args:
    a (int or float): Il primo numero.
    b (int or float): Il secondo numero.

Returns:
    int or float: La somma dei due numeri.
"""
return a + b
```

### 5.3 Funzione built-in help()

La funzione help() può essere usata per ottenere informazioni su classi, funzioni e moduli. Ad esempio:

```
help('stringa'.isalpha)
help(somma)
```

### 5.4 Funzioni built-in

Alcune funzioni built-in di Python:

```
len()    # Ritorna la lunghezza di un oggetto
max()    # Ritorna il valore massimo
min()    # Ritorna il valore minimo
sum()    # Ritorna la somma di una sequenza
type()   # Ritorna il tipo di un oggetto
```

## 6 Scope

Python segue la regola LEGB per risolvere il riferimento a una variabile:

- **Local**: le variabili definite all'interno di una funzione

- **Enclosing**: le variabili definite nella funzione esterna (se una funzione è definita all'interno di un'altra)
- **Global**: le variabili definite a livello del modulo
- **Built-in**: i nomi predefiniti di Python

### 6.1 Linee guida per scrivere buone funzioni

Una buona funzione dovrebbe agire solo su variabili locali, ovvero i dati devono essere passati come argomenti:

```
def eleva_al_quadrato(numero):
risultato = numero*numero
return risultato
```

Una buona funzione dovrebbe sempre tornare il risultato con un'istruzione return, invece di modificare direttamente gli argomenti:

```
def eleva_al_quadrato(numero):
risultato = numero*numero
return risultato
```

## 7 Moduli

Un modulo è un file che contiene definizioni di funzioni, classi e variabili. Per importare un modulo si usa la parola chiave `import`.

```
import math
var = math.sqrt(4)  # var = 2
Alternativa
from math import sqrt
var = sqrt(4)  # var = 2
```

## 8 Strutture Dati

### 8.1 Liste

Le liste sono sequenze mutabili di oggetti. Possono contenere elementi di tipi diversi.

```
mia_lista = [1, 2, 3]              # Lista di numeri
mia_lista = ['Marco', 'Irene', 'Paolo'] # Lista di
    stringhe
mia_lista = [15, 32, 'ambo']       # Lista eterogenea
```

### 8.1.1 Operatori di appartenenza

```
x in y      # True se x    nella lista y
x not in y  # True se x non    nella lista y
```

### 8.1.2 Accesso per posizione

```
mia_lista[0]   # Primo elemento
mia_lista[1]   # Secondo elemento
mia_lista[-1]  # Ultimo elemento
```

### 8.1.3 Slicing

```
mia_lista[1:6:2]  # Dall'elemento di indice 1 all'
   elemento di indice 6 (escluso), con step 2
mia_lista[::2]    # Dall'inizio alla fine, con step 2
```

### 8.1.4 Mutabilità delle liste

Le liste sono mutabili:

```
x[1:3] = [1,2,3]  # Sostituisce gli elementi dall'indice
   1 all'indice 3 (escluso)
x[1:3] = []       # Elimina gli elementi dall'indice 1
   all'indice 3 (escluso)
```

### 8.1.5 Differenze tra liste e stringhe

Le stringhe sono immutabili, mentre i metodi delle liste modificano la lista stessa.

## 8.2 Tuple

Le tuple sono sequenze immutabili di oggetti, definite usando le parentesi tonde. Possono essere usate per assegnare più valori contemporaneamente:

```
tupla1 = (2,)
tupla2 = 2,
(a,b,c) = (1,2,3)
```

## 8.3 Dizionari

I dizionari associano valori (value) a chiavi (key).

```
mio_diz = {'Trieste': 34100, 'Padova': 35100}  # Dizionario di numeri
mio_diz = {'Trieste': 'TS', 'Padova': 'PD'}    # Dizionario di stringhe
Accesso al valore tramite chiave
sigla_ts = mio_diz['Trieste']  # sigla_ts = 'TS'
Modifica del dizionario
mio_diz['Venezia'] = 30121     # Aggiunge un nuovo elemento
mio_diz['Venezia'] = 30100     # Modifica un valore esistente
del mio_diz['Padova']          # Rimuove un elemento
```

### 8.4 Set

I set sono collezioni non ordinate di oggetti unici e immutabili. Supportano operazioni come unione, intersezione e differenza:

```
a = {'a', 'b', 'c'}
b = set('abracadabra')
print(a - b)  # differenza
print(a | b)  # unione
print(a & b)  # intersezione
print(a ^ b)  # differenza simmetrica
```

## 9 Essere Pythonici

Essere "pythonici" significa sfruttare le potenzialità del linguaggio per scrivere codice più leggibile ed efficiente.

### 9.1 Modo non pythonico

```
mia_lista = ['a', 'b', 'c']
i = 0
while (i < len(mia_lista)):
print(mia_lista[i])
i = i + 1
Controllare se un elemento    presente in una lista
for i in range(len(mia_lista)):
if mia_lista[i] == "a":
print("Trovato 'a'!")
```

### 9.2 Modo pythonico

Iterare sugli elementi di una lista in modo "pythonico":

```
mia_lista = ['a', 'b', 'c']
for elemento in mia_lista:
print(elemento)
Controllare se un elemento    presente in una lista
if "a" in mia_lista:
print("Trovato 'a'!")
```

# 10 File e CSV

## 10.1 Operazioni sui file

```
Apertura di un file
mio_file = open('shampoo_sales.csv', 'r')  # 'r' =
    modalit  lettura
Leggere i primi 50 caratteri
print(mio_file.read()[0:50])
Chiudere il file
mio_file.close()
```

```
Leggere le prime 5 righe
mio_file = open('shampoo_sales.csv', 'r')
for i in range(5):
print(mio_file.readline())
mio_file.close()
```

```
Modo pythonico di leggere un file
with open('shampoo_sales.csv', 'r') as mio_file:
for linea in mio_file:
print(linea)
Il file viene chiuso automaticamente
```

## 10.2 Leggere un file CSV

```
Funzione split per dividere una stringa
mia_stringa = 'Ciao mondo'
pezzi = mia_stringa.split(' ')  # pezzi = ['Ciao', 'mondo
    ']
```

## 10.3 Casting

```
float('9.1')  # Da stringa numerica a floating point: 9.1
int(9.1)      # Da floating point a intero: 9
float(9)      # Da intero a floating point: 9.0
str(9)        # Da intero a stringa: '9'
```

```
Lettura di un file CSV
dati = []
with open('shampoo_sales.csv', 'r') as mio_file:
for linea in mio_file:
elementi = linea.split(',')
if elementi[0] != 'Date':  # Salta l'intestazione
data = elementi[0]
valore = float(elementi[1])
dati.append([data, valore])
```

# 11 Programmazione Orientata agli Oggetti

## 11.1 Classi e oggetti

```
class Persona:
def init(self, nome, cognome):
self.nome = nome
self.cognome = cognome
def saluta(self):
    print("Ciao, sono {} {}".format(self.nome, self.
   cognome))
Istanziazione
persona1 = Persona('Mario', 'Rossi')
persona2 = Persona('Irene', 'Bianchi')
print(persona1.nome)  # 'Mario'
persona1.saluta()     # 'Ciao, sono Mario Rossi'
```

```
Liste di oggetti
persone = [Persona('Mario', 'Rossi'),
Persona('Irene', 'Bianchi')]
print(persone[0].nome)  # Mario
persone[1].saluta()     # 'Ciao, sono Irene Bianchi'
```

## 11.2 Ereditarietà

```
class Studente(Persona):
# Sovrascrittura del metodo saluta
def saluta(self):
print("Ciao, sono {} e sono uno studente.".format(self.
   nome))
```

```
class Professore(Persona):
def str(self):
return 'Prof. "{} {}"'.format(self.nome, self.cognome)
# Sovrascrittura del metodo saluta
def saluta(self):
```

```
    print('Ciao, sono il Prof. {} {}'.format(self.nome,
    self.cognome))

# Utilizzo del metodo originale tramite super()
def saluta_informale(self):
    super().saluta()
```

### 11.3 Ispezione degli oggetti

```
isinstance(variabile, MiaClasse)  # Verifica se
    istanza di una classe
issubclass(MiaClasse1, MiaClasse2)  # Verifica se
    sottoclasse
dir(oggetto)  # Elenca attributi e metodi dell'oggetto
type(oggetto)  # Ritorna la classe dell'oggetto
```

## 12 Gestione degli Errori

### 12.1 Eccezioni

```
Try-except basilare
mia_var = 'ciao'
try:
mia_var = float(mia_var)
except:
print('Non posso convertire "mia_var" a numero!')
print('Uso il valore di default di "0.0"')
mia_var = 0.0
print(mia_var)  # Stampa 0.0
```

```
Try-except con gestione specifica delle eccezioni
try:
mia_var = float(mia_var)
except ValueError:
print('Errore di VALORE, "mia_var" valeva "{}".'.format(
    mia_var))
except TypeError:
print('Errore di TIPO, "mia_var" era "{}".'.format(type(
    mia_var)))
except Exception as error:
print('Errore generico: "{}".'.format(error))
```

```
Try-except-else-finally
try:
file_parametro = open(nome_file_parametro)
parametro_come_stringa = file_parametro.read()
parametro_come_float = float(parametro_come_stringa)
```

```
except IOError:
print('Non posso leggere il file!')
except ValueError:
print('Non posso convertire il parametro a valore
    numerico!')
else:
parametro = parametro_come_float  # Eseguito solo se non
    ci sono errori
finally:
file_parametro.close()  # Eseguito sempre
```

### 12.2 Generare eccezioni

```
parametro = -5
if parametro < 0:
raise ValueError('Il parametro    minore di zero')
```

```
Definizione di un'eccezione personalizzata
class InvalidParameter(Exception):
pass
parametro = -5
if parametro < 0:
raise InvalidParameter('Il parametro    minore di zero')
```

## 13 Controllo degli Input

### 13.1 Verifica del tipo

```
def somma_lista(lista_input):
if not isinstance(lista_input, list):
raise TypeError('Il tipo di lista_input non    lista, ' +
'ma "{}"'.format(type(lista_input)))
somma = 0
for elemento in lista_input:
somma += elemento
return somma
```

```
Controllo anche del tipo degli elementi
def somma_lista(lista_input):
if not isinstance(lista_input, list):
raise TypeError('Il tipo di lista_input non    lista')
for i, element in enumerate(lista_input):
if not (isinstance(element, int) or isinstance(element,
    float)):
raise TypeError('Il tipo di elemento in posizione {} '.
    format(i) +
'non    n   intero  n   float')
```

```
somma = 0
for elemento in lista_input:
somma += elemento
return somma
```

### 13.2 Sanitizzazione degli input

```
Sanitizzazione di input
sesso = sesso.upper()  # Trasforma in maiuscolo
sesso = sesso.strip()  # Rimuove spazi iniziali e finali
if sesso not in ['M', 'F']:
raise ValueError('Il valore del sesso non    n  M n  F'
    )
```

## 14 Testing

### 14.1 Test semplice

```
def somma(a, b):
return a + b
Testing
if not somma(1, 1) == 2:
raise Exception('Test 1+1 non passato')
if not somma(1.5, 2.5) == 4:
raise Exception('Test 1.5+2.5 non passato')
```

### 14.2 Test con unittest

```
from unittest import TestCase
from somma import somma
class TestSomma(TestCase):
def test_somma(self):
self.assertEqual(somma(1, 1), 2)
self.assertEqual(somma(1.5, 2.5), 4)
```

## 15 Modelli Predittivi

### 15.1 Modello semplice

```
class TrendModel():
def predict(self, data):
if len(data) != 3:
raise ValueError('Passati {} mesi'.format(len(data)) +
' ma il modello richiede di averne 3.')
```

```
variations = []
item_prev = None
for item in data:
if item_prev is not None:
variations.append(item-item_prev)
item_prev = item
avg_variation = sum(variations)/len(variations)
prediction = data[-1] + avg_variation
return prediction
```

### 15.2 Test del modello

```
from models import TrendModel
Definisco dei dati di test
test_data = [50, 52, 60]
Istanzio il modello
trend_model = TrendModel()
Applico il modello
prediction = trend_model.predict(test_data)
Test
if prediction != 65:
raise Exception('Il modello sbaglia la prediction')
Test con dati non validi
wrong_test_data = [50, 52, 60, 70]
try:
trend_model.predict(wrong_test_data)
except ValueError:
pass
else:
raise Exception('Il modello non alza eccezioni con dati
    errati')
print('ALL TESTS PASS')
```

### 15.3 Modello parametrizzato

```
class TrendModel():
def init(self, window=3):
self.window = window
def predict(self, data):
    if len(data) != self.window:
        raise ValueError('Passati {} mesi'.format(len(
    data)) +
                         ' ma il modello richiede di' +
                         ' averne {}.'.format(self.window))
    variations = []
    item_prev = None
    for item in data:
```

```
        if item_prev is not None:
            variations.append(item-item_prev)
        item_prev = item
    avg_variation = sum(variations)/len(variations)
    prediction = data[-1] + avg_variation
    return prediction
```

### 15.4 Modello con fit

```
class Model():
def fit(self, data):
# Fit non implementato nella classe base
raise NotImplementedError('Metodo non implementato')
def predict(self, data):
    # Predict non implementato nella classe base
    raise NotImplementedError('Metodo non implementato')
class TrendModel(Model):
def init(self, window=3):
self.window = window
def validate_data(self, data):
    if len(data) != self.window:
        raise ValueError('Passati {} mesi'.format(len(
   data)) +
                         ' ma il modello richiede di' +
                         ' averne {}.'.format(self.window))

def compute_avg_variation(self, data):
    variations = []
    item_prev = None
    for item in data:
        if item_prev is not None:
            variations.append(item-item_prev)
        item_prev = item
    avg_variation = sum(variations)/len(variations)
    return avg_variation

def predict(self, data):
    self.validate_data(data)
    prediction = data[-1] + self.compute_avg_variation(
   data)
    return prediction
class FitTrendModel(TrendModel):
def fit(self, data):
self.historical_avg_variation = self.
   compute_avg_variation(data)
def predict(self, data):
    try:
        self.historical_avg_variation
    except AttributeError:
```

```
38          raise Exception('Il modello non    fittato!')
39      self.validate_data(data)
40      prediction = data[-1] + (self.
    historical_avg_variation +
41                               self.compute_avg_variation(
    data))/2
42      return prediction
```

### 15.5 Valutazione del modello

```
1  def evaluate(self, data):
2  # Cutoff per dividere i dati (70% fit, 30% test)
3  fit_data_cutoff = int(len(data)*0.7)
4  # Divido i dati
5  fit_data = data[:fit_data_cutoff]
6  test_data = data[fit_data_cutoff:]
7
8  # Provo a fare il fit
9  try:
10     self.fit(fit_data)
11 except Exception as e:
12     if isinstance(e, NotImplementedError):
13         pass
14     else:
15         raise Exception('Il metodo fit    implementato ma
       ' +
16                         'ha sollevato una eccezione {}'.
       format(e))
17
18 # Calcolo MAE (Mean Absolute Error)
19 total_error = 0
20 n_predictions = 0
21
22 # Per ogni finestra di dati nel test set
23 for i in range(len(test_data) - self.window):Di seguito
       il resto del documento .tex con le integrazioni
       suggerite:
24     # Prendo una finestra
25     window_data = test_data[i:i+self.window]
26     # Faccio la previsione
27     prediction = self.predict(window_data)
28     # Il valore vero    quello subito dopo la finestra
29     true_value = test_data[i+self.window]
30     # Calcolo errore assoluto
31     error = abs(prediction - true_value)
32     total_error += error
33     n_predictions += 1
34
35 # Calcolo MAE finale
```

```
if n_predictions > 0:
    evaluation_mae = total_error / n_predictions
else:
    evaluation_mae = None

return evaluation_mae
```