ITS TECNOLOGIE DELL'INFORMAZIONE E DELLA COMUNICAZIONE

NOME CORSO

Unità Formativa (UF): AI e Machine Learning - Python

Docente: Fabio Giuseppe Antonio Gagliardi

Titolo argomento: Basi di Python

python_cheat_sheet

September 9, 2022

0.1 Tipi base

[ ]:
```
# ============================
# interpretati automaticamente
# ============================
dint = -4
dfloat = 32.0
dstring = "ciao"

print("tipi interpretati")
print(dint, "=", type(dint))
print(dfloat, "=", type(dfloat))
print(dstring, "=", type(dstring))
print("\n")


# =========================
# dichiarati esplicitamente
# =========================

ex_dint: int = 32
ex_dfloat: float = 34.0
ex_dstring: str = "ciao"

print("tipi espliciti")
print(ex_dint, "=", type(ex_dint))
print(ex_dfloat, "=", type(ex_dfloat))
print(ex_dstring, "=", type(ex_dstring))
print("\n")
```

0.2 Operazioni con le stringhe

[ ]:
```
# ============================
# come dichiarare una stringa
# ============================

str_1 = "quseta è una stringa"
```

```
str_2 = 'una nuova stringa'
testo = '''
  esempio con qualsiasi tipo di carattere:
  ' ""
 '''
print("\n")

print("dichiarazioni di stringhe")
print(str_1)
print(str_2)
print(testo)
print("\n")


# ====================
# concatenare stringhe
# ====================

str_cn = str_1 + " più " + str_2
print("stringhe concatenate")
print(str_cn)
print("\n")
```

1 Operazioni Matematiche

[ ]:
```
# =========
# esponente
# =========
p2 = 3 ** 2
p3 = 3 ** 3
print("esponenti 3^2, 3^3: ", p2, p3)
print("\n")

# =====
# resto
# =====
mod = 17 % 2
print("resto di 17 / 2: ", mod)
print("\n")

# ================
# divisione interi
# ================
di = 17 // 2
```

```
df = 17 / 2
print("divisiioni di interi e virgola mobile 17 / 2: ", di, df)
print("\n")
```

2 Funzioni

[ ]:
```
# ======================
# dicharare una funzione
# ======================
def stampa_qualcosa():
  print("ciao!")
print("\n")


# =================
# parametri formali
# =================
def transla_posizione(x, y, z):
  print(x, type(x), y, type(y), z, type(z))

def ruota(theta: float = 30, phi: float = 20):
    print(theta, type(theta), phi, type(phi))
print("\n")




# ===============================
# tipo restituito da una funzione
# ===============================
def foo() -> float:
  return 32.0



# ============================================================
# una funzione in python può restituire più di una variabile
# ============================================================
def baar():
  return 32, "ciao"

# assegna i valori nello stesso ordine in cui vengono restituiti
vi, vs = baar()
```

3 Liste

[ ]:
```python
from typing import List




# ====================
# come creare le liste
# ====================
numeri : list = [1, 2, 3, 4, 5, 6]
stringhe = ["paorla 1", "parola 2", "parola 3"]
misto = [32, "trentadue", 40, "quaranta"]
booleani = [True, False, True, False]

lista_annidata = [[32, 34, 532],["a", "b", 12, 3]]



# ======================
# combinazioni di liste
# ====================
num_str = numeri + stringhe
print("combinazioni di liste: ")
print(num_str)
print("\n")




# =================
# accesso e slicing
# =================
lista = [32, 12, 16, 4, 16, 2, 5, 6, 7, 15]
print("accesso via indice: ")
print(lista[3])
print("accesso via slicing: ")
print(lista[1:3])
print("\n")



# ===========
# ordinamento
# ===========
```

```python
# !!!!!!!!!!!
# Attenzione!
# !!!!!!!!!!!
# Il segno "=" non copia una struttura dati, ma passa un riferimento a␣
 ↪qeust'ultima.
# Per non cambiare il contento di una struttura dati durante un operazione sui
# dati conenuti, va quindi creata esplicitamente una nuova copia.
# Nel caso seguente cambiando la variabile "ref" cambia anche la lista a cui fa␣
 ↪riferimento
#␣
 ↪=============================================================================================

ref = lista
cpy = lista.copy()

ref.sort()
print("lista orginale: ")
print(lista)
print("lista con riferimento all'originale: ")
print(ref)
print("lista copiata dall'originale: ")
print(cpy)
print("\n")




# =====================
# Comprensione di lista
# =====================
cm1 = [x for x in cpy if x > 3]
print("lista generata itrando su una lista esistente: ")
print(cm1)
print("lista generata creando vaolori durante il ciclo for (vedere generatori):␣
 ↪")
cm2 = [x ** 2 for x in range(3, 12)]
print(cm2)
print("\n")


# ========================================================================
# len() funzione built-in per contare gli elementi di una struttura dati
# ========================================================================
nl = len(lista)
```

```
# ================================================================================
# filter() funzione built-in per filtrare una struttura dai con una funzione
# ================================================================================
def filtro(t):
  if t > 5:
    return True
  else:
    return False

check = filter(filtro, cpy)
print("dal filtro alla lista")
print(type(check))
print(list(check))
print("\n")
```

4 Tuple

[ ]:
```
# ====================================================================
# struttura dati simile alla lista, tranne per il fatto
# che sia immutabile e i valori contenuti non possono essere cambiati
# ====================================================================
tupla = (1, 2, 3, 4, 5, 6)
```

5 Generatori e “yield”

[ ]:
```
#␣
 ↪==========================================================================================
# un "generatore" è un costrutto di python che permette di generare valori "on␣
 ↪the fly".
# L'operatore non conserva alcun valore in memoria, quest'ultimo viene creato e
# subito dopo scartato. Si potrebbe dire che conserva il modo in cui un dato␣
 ↪numero
# di variabili viene generato.
#␣
 ↪==========================================================================================


# creazione di un generatore
generator = (x * x for x in range(5))

# uso di un generatore
print("generatore: ")
for i in generator:
  print(i)
```

```
print("\n")


#␣
 ↪==========================================================================================
# yield é una parola chiave simile a "return", la differza sta nel fatto che
# ritorni un oggetto "generatore"
#␣
 ↪==========================================================================================

def crea_un_generatore():
  for i in range(5):
    yield i + i

generator_fun = crea_un_generatore()
print("yield: ")
for i in generator_fun:
  print(i)
print("\n")
```

6 Dizionari

[ ]:
```
from typing import Dict


# =========================
# come creare un dizionario
# key : value
# =========================
auto_prezzi : dict = {
     "fiat 500l"  : 15000,
     "fiat 126"   : 10000,
     "fiat 127"   : 40000,
     "fiat panda" : 30000
}




# ==================
# accssso per chiave
# ==================
print("prezzo fiat 500l: ", auto_prezzi["fiat 500l"])
print("\n")
```

```
# stampa tutte le chiavi
for x in auto_prezzi:
  print(x)
print("\n")

# stampa tutti i valori
for x in auto_prezzi:
  print(auto_prezzi[x])
print("\n")

# stampa chiavi e valori
for x, y in auto_prezzi.items():
  print(x, y)
print("\n")
```

7 Istruzioni di selezione e operatori logici

[ ]:
```
a = 1
b = 4

if a < b:
  print("[1] a < b")



if a != b:
  if a < b:
    print("[2] a < b")



if (a != b) and (a < b):
    print("[3] a < b")



if (a < b) and not(a == b):
    print("[4] a < b")

c = -4
if (a < c) or (a < b):
    print("[5] a < b")


if a == b:
  print("a == b")
```

```
elif a < b:
  print("[6] a < b")



if a == b:
  print("a == b")
elif a > b:
  print("a > b")
else:
  print("[7] a < b")
```

8 Cicli

[ ]:
```
# ========================================
# stampa dei singoli caratteri di una stringa
# ========================================
for x in "python":
  print(x)
print("\n")


# ================================================
# stampa numeri fino a quando "i" resta minore di 10
# ================================================
i = 0
while i < 10:
  print(i)
  i += 1
print("\n")


# ============================================================================
# stampa numeri fino a quando "i" resta minore di 10 ma se "i" è uguale a 3
# interrompe il ciclo subito dopo
# ============================================================================
i = 0
while i < 4:
  print(i)
  if i == 3:
    break
  i += 1
print("\n")
```

9 Classe

[ ]:
```
# =================
# creare una classe
# =================
class Test1:
  x = 5

t1 = Test1()
print("valore membro classe Test1: ", t1.x)




# ========================================
# creare una classe con costrutture e metodi
# ========================================
class Test2():
  def __init__(self, x: float, y: float, z: float):
    self.x : float = x
    self.y : float = y
    self.z : float = z

  def incrementa(self, x: float, y: float, z: float):
    self.x += x
    self.y += y
    self.z += z

  def decrementa(self, x: float, y: float, z: float):
    self.x -= x
    self.y -= y
    self.z -= z

t2 = Test2(32, 15, 43)
print("Valori Test2: ", t2.x, t2.y, t2.z)
t2.incrementa(1, 2, 3)
print("Valori Test2 incrementati: ", t2.x, t2.y, t2.z)
t2.decrementa(10, 22, 41)
print("Valori Test2 decrementati: ", t2.x, t2.y, t2.z)
print("\n")



# ============
# Ereditarietà
# ============
class Test3(Test2):
```

```
  def __init__(self, x: float, y: float, z: float, w: float):
    super().__init__(x, y, z)
    self.w = w

  def incrementa(self, x: float, y: float, z: float, w: float):
    return super().incrementa(x, y, z)
    self.w += w

  def decrementa(self, x: float, y: float, z: float, w: float):
    return super().decrementa(x, y, z)
    self.w -= w

t3 = Test3(23, 54, 22, 15)
print("Valori Test3: ", t3.x, t3.y, t3.z)
t3.incrementa(1, 2, 3, 4)
print("Valori Test3 incrementati: ", t3.x, t3.y, t3.z, t3.w)
t3.decrementa(10, 22, 41, 16)
print("Valori Test3 decrementati: ", t3.x, t3.y, t3.z, t3.w)
```