Corso ITS: *ARTIFICIAL INTELLIGENCE SPECIALIST*

Modulo: **Programmazione ad oggetti in Python e librerie esterne**

Docente: *Andrea Ribuoli*

Martedì **25 Febbraio** 2025

09:00 - 14:00

In [1]:
```
type(""), type(()), type([]), type(set()), type({})
```

Out[1]:
```
(str, tuple, list, set, dict)
```

- completiamo l'elenco delle classi *contenitore* fornite a standard da Python
- le **tuple**: sono liste *immutabili*

In [2]:
```
b = (2, "Hamburger", 2.8)
type(b)
```

Out[2]:
```
tuple
```

In [3]:
```
b[1]
```

Out[3]:
```
'Hamburger'
```

In [4]:
```
b[2]=3.1
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[4], line 1
----> 1 b[2]=3.1

TypeError: 'tuple' object does not support item assignment
```

- dunque la possibilità di selezionare tramite indice
- non va di pari passo con la possibilità di aggiornare tramite indice
- ossia l'**item assignment**
- un altro caso di supporto della selezione con indice ma non dell'*item assignment* ...

In [ ]:
```
s = "Giovedi"
s[6]
```

Out[ ]:
```
'i'
```

In [ ]:
```
s = "Giovedi"
s[6] = "ì"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [6], in <cell line: 2>()
      1 s = "Giovedi"
----> 2 s[6] = "ì"

TypeError: 'str' object does not support item assignment
```

- passando attraverso una lista...

In [ ]:
```
s = "Giovedi"
l = list(s)
l[6] = "ì"
s = "".join(l)
s
```

Out[ ]:
```
'Giovedì'
```

---

ereditarietà

- uso di `super().__init__(...)` per invocare il costruttore della classe padre
- la invocazione di *__init__* è **senza** il passaggio di *self*
- con l'**overloading** posso differenziare il comportamento
- ad esempio in stampa: **__repr__**

In [ ]:
```
class Persona :
    def __init__(self, nome, cognome) :
        self._nome = nome
        self._cognome = cognome
        self._age = 0
    def setName(self, nome) :
        self._nome = nome
    def setSurname(self, cognome) :
        self._cognome = cognome
    def setAge(self, eta) :
        self._age = eta
    def __repr__(self) :
        return self._nome + " " + self._cognome

class Docente(Persona) :
    def __init__(self, nome, cognome, materia) :
```

```
        super().__init__(nome, cognome)
        self._topic = materia
    def __repr__(self) :
        return self._nome + " " + self._cognome + " insegna nel modulo di "

class Student(Persona) :
    def __init__(self, nome, cognome, annoDiCorso) :
        super().__init__(nome, cognome)
        self._year = annoDiCorso
    def __repr__(self) :
        return self._nome + " " + self._cognome + " frequenta l'anno " + str
```

In [ ]:
```
andrea = Docente("Andrea", "Ribuoli", "Python")
print(andrea)
andrea.setAge(62)
olesia = Student("Olesia", "Rudenko", 1)
print(olesia)
olesia.setAge(36)
giulia = Persona("Giulia", "Dezi")
print(giulia)
```

```
Andrea Ribuoli insegna nel modulo di Python
Olesia Rudenko frequenta l'anno 1
Giulia Dezi
```

DataFrame

- molti oggetti di uso comune sono implementati come istanze di classi addizionali
- fornite da moduli ormai moto noti
- un esempio è la classe **DataFrame** offerta da **pandas**
- un *dataframe* è una struttura dati atta a memorizzare dati tabulari
- è resa disponibile dal pacchetto **Pandas**

In [ ]:
```
import pandas as pd
df = pd.DataFrame([['Andrea', 'Ribuoli', 62],
                   ['Laura', 'Fusaro', 59],
                   ['Roberto', 'Ribuoli', 35],
                   ['Giovanni', 'Ribuoli', 29],
                   ['Francesca', 'Ribuoli', 23]])
```

In [ ]:
```
type(df)
```

Out[ ]:
```
pandas.core.frame.DataFrame
```

In [ ]:
```
metodi = dir(df)
for metodo in metodi :
    if not metodo.startswith('_') and metodo.startswith('col') :
        print(metodo)
```

```
columns
```

- possiamo passare a columns un oggetto di tipo **list** con i nomi delle colonne

In [ ]:
```
df.columns = ["name", "surname", "age"]
```

- all'interno di **Jupyter Notebook** la resa grafica è molto elegante

In [ ]:
```
df
```

Out[ ]:

	name	surname	age
0	Andrea	Ribuoli	62
1	Laura	Fusaro	59
2	Roberto	Ribuoli	35
3	Giovanni	Ribuoli	29
4	Francesca	Ribuoli	23

- anche la resa in stampa è gradevole:

In [ ]:
```
print(df)
```

```
        name  surname  age
0     Andrea  Ribuoli   62
1      Laura   Fusaro   59
2    Roberto  Ribuoli   35
3   Giovanni  Ribuoli   29
4  Francesca  Ribuoli   23
```

- posso utilizzare **iloc** e selezionare righe e colonne per range di indici

In [ ]:
```
df.iloc[2:4,1:3]
```

Out[ ]:

	surname	age
2	Ribuoli	35
3	Ribuoli	29

- oppure posso utilizzare i titoli delle colonne

In [ ]:
```
df[["surname", "age"]]
```

Out[ ]:

	surname	age
0	Ribuoli	62
1	Fusaro	59
2	Ribuoli	35
3	Ribuoli	29
4	Ribuoli	23

- e successivamente filtrare in base ai valori

In [ ]:
```
df[["surname", "age"]][(df.age > 25) & (df.age < 40)]
```

Out[ ]:

	surname	age
2	Ribuoli	35
3	Ribuoli	29

---

persistenza

salva.py

In [ ]:
```
import pickle
import pandas
df = pandas.DataFrame([['Andrea', 'Ribuoli', 62],
                       ['Laura', 'Fusaro', 59],
                       ['Roberto', 'Ribuoli', 35],
                       ['Giovanni', 'Ribuoli', 29],
                       ['Francesca', 'Ribuoli', 23]])
df.columns = ["nome", "cognome", "eta"]
dbfile = open('status', 'wb')
pickle.dump(df, open('status', 'wb'))
dbfile.close()
```

leggi.py

In [ ]:
```
import pickle
import pandas
dbfile = open('status', 'rb')
df = pickle.load(dbfile)
print(df[df.eta < 40])
dbfile.close()
```

```
        nome  cognome  eta
2    Roberto  Ribuoli   35
3   Giovanni  Ribuoli   29
4  Francesca  Ribuoli   23
```

---

urllib e BeatifulSoup

In [ ]:
```
import urllib.request
address ='https://www.andrearibuoli.it'
response = urllib.request.urlopen(address)
theBytes = response.read()
html = theBytes.decode()
print(len(html))
```

```
4135
```

In [ ]:
```
from bs4 import BeautifulSoup
doc = BeautifulSoup(html)
print(doc.title.string)
```

```
           www.andrearibuoli.it
```

In [ ]:
```
print(doc.title.parent.name)
```

```
head
```

In [ ]:
```
links = doc.find_all('a')
for link in links :
    print(link.get('href'))
```

```
https://www.andrearibuoli.it/wp
https://it.linkedin.com/in/andrea-ribuoli-5b37403b
https://github.com/AndreaRibuoli
https://www.andrearibuoli.it/Meeting/PU_30_11_2024.zip
https://www.andrearibuoli.it/Meeting/PU_20_4_2024.zip
https://www.andrearibuoli.it/Meeting/ITPASS_11_novembre_2023.tar.gz
https://www.andrearibuoli.it/Meeting/ITPASS_6_maggio_2023.tar.gz
https://www.andrearibuoli.it/Meeting/Meeting%20ITPASS%2016%20Ottobre%20202
1.zip
```

---

caso di studio

studio

In [ ]:
```
import urllib.request
address ='https://www.comuni-italiani.it/province.html'
response = urllib.request.urlopen(address)
theBytes = response.read()
html = theBytes.decode()
print(len(html))
```

```
---------------------------------------------------------------------------
UnicodeDecodeError                        Traceback (most recent call last)
Input In [4], in <cell line: 5>()
      3 response = urllib.request.urlopen(address)
      4 theBytes = response.read()
----> 5 html = theBytes.decode()
      6 print(len(html))

UnicodeDecodeError: 'utf-8' codec can't decode byte 0xe0 in position 202: in
valid continuation byte
```

In [ ]:
```
import urllib.request
address ='https://www.comuni-italiani.it/province.html'
response = urllib.request.urlopen(address)
theBytes = response.read()
html = theBytes.decode(encoding='iso-8859-1')
print(len(html))
```

```
44896
```

In [ ]:
```
from bs4 import BeautifulSoup
t1 = BeautifulSoup(html, "html.parser").table
t2 = t1.find_next("table")
t3 = t2.find_next("table")
t4 = t3.find_next("table")
riga = t4.find_next("tr")
riga = riga.find_next("tr")
province = {}
while riga != None :
    tdx  = riga.find_next("td")
    tdx  = tdx.find_next("td")
    prov = tdx.get_text()
    tdx  = tdx.find_next("td")
    tdx  = tdx.find_next("td")
    tdx  = tdx.find_next("td")
    tdx  = tdx.find_next("td")
    tdx  = tdx.find_next("td")
    tdx  = tdx.find_next("td")
    sigla = tdx.get_text()
    province[sigla]=prov
    if sigla == 'VT' : break
    riga = riga.find_next("tr")
province
```

Out[ ]:
```
{'AG': 'Agrigento',
 'AL': 'Alessandria',
 'AN': 'Ancona',
 'AO': 'Aosta',
 'AR': 'Arezzo',
 'AP': 'Ascoli Piceno',
 'AT': 'Asti',
 'AV': 'Avellino',
 'BA': 'Bari',
 'BT': 'Barletta-Andria-Trani',
 'BL': 'Belluno',
 'BN': 'Benevento',
 'BG': 'Bergamo',
 'BI': 'Biella',
 'BO': 'Bologna',
 'BZ': 'Bolzano',
 'BS': 'Brescia',
 'BR': 'Brindisi',
 'CA': 'Cagliari',
 'CL': 'Caltanissetta',
 'CB': 'Campobasso',
 'CI': 'Carbonia-Iglesias',
 'CE': 'Caserta',
 'CT': 'Catania',
 'CZ': 'Catanzaro',
 'CH': 'Chieti',
 'CO': 'Como',
 'CS': 'Cosenza',
 'CR': 'Cremona',
 'KR': 'Crotone',
 'CN': 'Cuneo',
 'EN': 'Enna',
 'FM': 'Fermo',
 'FE': 'Ferrara',
 'FI': 'Firenze',
 'FG': 'Foggia',
 'FC': 'Forlì-Cesena',
 'FR': 'Frosinone',
 'GE': 'Genova',
 'GO': 'Gorizia',
 'GR': 'Grosseto',
 'IM': 'Imperia',
 'IS': 'Isernia',
 'SP': 'La Spezia',
 'AQ': "L'Aquila",
 'LT': 'Latina',
 'LE': 'Lecce',
 'LC': 'Lecco',
 'LI': 'Livorno',
 'LO': 'Lodi',
 'LU': 'Lucca',
 'MC': 'Macerata',
 'MN': 'Mantova',
 'MS': 'Massa-Carrara',
 'MT': 'Matera',
 'ME': 'Messina',
```

```
 'MI': 'Milano',
 'MO': 'Modena',
 'MB': 'Monza e della Brianza',
 'NA': 'Napoli',
 'NO': 'Novara',
 'NU': 'Nuoro',
 'OT': 'Olbia-Tempio',
 'OR': 'Oristano',
 'PD': 'Padova',
 'PA': 'Palermo',
 'PR': 'Parma',
 'PV': 'Pavia',
 'PG': 'Perugia',
 'PU': 'Pesaro e Urbino',
 'PE': 'Pescara',
 'PC': 'Piacenza',
 'PI': 'Pisa',
 'PT': 'Pistoia',
 'PN': 'Pordenone',
 'PZ': 'Potenza',
 'PO': 'Prato',
 'RG': 'Ragusa',
 'RA': 'Ravenna',
 'RC': 'Reggio Calabria',
 'RE': 'Reggio Emilia',
 'RI': 'Rieti',
 'RN': 'Rimini',
 'RM': 'Roma',
 'RO': 'Rovigo',
 'SA': 'Salerno',
 'VS': 'Medio Campidano',
 'SS': 'Sassari',
 'SV': 'Savona',
 'SI': 'Siena',
 'SR': 'Siracusa',
 'SO': 'Sondrio',
 'TA': 'Taranto',
 'TE': 'Teramo',
 'TR': 'Terni',
 'TO': 'Torino',
 'OG': 'Ogliastra',
 'TP': 'Trapani',
 'TN': 'Trento',
 'TV': 'Treviso',
 'TS': 'Trieste',
 'UD': 'Udine',
 'VA': 'Varese',
 'VE': 'Venezia',
 'VB': 'Verbano-Cusio-Ossola',
 'VC': 'Vercelli',
 'VR': 'Verona',
 'VV': 'Vibo Valentia',
 'VI': 'Vicenza',
 'VT': 'Viterbo'}
```

In [ ]:
```
from bs4 import BeautifulSoup
```

```
t1 = BeautifulSoup(html, "html.parser").table
t2 = t1.find_next("table")
t3 = t2.find_next("table")
t4 = t3.find_next("table")
riga = t4.find_next("tr")
riga = riga.find_next("tr")
import pandas as pd
province = pd.DataFrame(columns = ["sigla", "nome"])
i = 0
while riga != None :
    tdx  = riga.find_next("td")
    tdx  = tdx.find_next("td")
    prov = tdx.get_text()
    tdx  = tdx.find_next("td")
    tdx  = tdx.find_next("td")
    tdx  = tdx.find_next("td")
    tdx  = tdx.find_next("td")
    tdx  = tdx.find_next("td")
    tdx  = tdx.find_next("td")
    sigla = tdx.get_text()
    province.loc[i] = [sigla, prov]
    if sigla == 'VT' : break
    i += 1
    riga = riga.find_next("tr")
province
```

Out[ ]:

	sigla	nome
0	AG	Agrigento
1	AL	Alessandria
2	AN	Ancona
3	AO	Aosta
4	AR	Arezzo
...	...	...
105	VC	Vercelli
106	VR	Verona
107	VV	Vibo Valentia
108	VI	Vicenza
109	VT	Viterbo

110 rows × 2 columns

In [ ]:
```
province[(province.sigla >= 'P') & (province.sigla <= 'PZ')]
```

Out[ ]:

	sigla	nome
64	PD	Padova
65	PA	Palermo
66	PR	Parma
67	PV	Pavia
68	PG	Perugia
69	PU	Pesaro e Urbino
70	PE	Pescara
71	PC	Piacenza
72	PI	Pisa
73	PT	Pistoia
74	PN	Pordenone
75	PZ	Potenza
76	PO	Prato