Corso ITS:

PROGETTISTA E SVILUPPATORE SOFTWARE:

FULL STACK DEVELOPER E CLOUD SPECIALIST

Modulo: **Programmazione in Python**

Docente: *Andrea Ribuoli*

Mercoledì **16 Aprile** 2025

09:00 - 14:00

---

esercizi sulla creazione di classi

In [1]:
```
class Numero :
    def __init__(self, n):
        self._valore = n
```

In [2]:
```
a = Numero(5)
```

In [5]:
```
b = Numero(7)
```

In [7]:
```
print(a, b, sep= "\n")
```

```
<__main__.Numero object at 0x70000000217d5b0>
<__main__.Numero object at 0x700000002182100>
```

In [8]:
```
print(a._valore, b._valore)
```

```
5 7
```

---

creiamo una classe **Stanza**

In [26]:
```
class Stanza:
    def __init__(self, dim1, dim2):
```

```
        self._ampiezza   = dim1
        self._profondita = dim2
    def ampiezza(self):
        return self._ampiezza
    def profondita(self):
        return self._profondita
    def area(self):
        return self._ampiezza * self._profondita

cucina = Stanza(4, 5)
bagno  = Stanza(3, 2)
salone = Stanza(4, 6)
print(cucina.profondita())
print(bagno.profondita())
print(salone.ampiezza())
print(salone.area())
## print(bagno._profondita)
```

```
5
2
4
24
```

In [27]:
```
class Stanza:
    def __init__(self, dim1, dim2):
        self._ampiezza   = dim1
        self._area       = dim1 * dim2
    def ampiezza(self):
        return self._ampiezza
    def profondita(self):
        return self._area // self._ampiezza
    def area(self):
        return self._area

cucina = Stanza(4, 5)
bagno  = Stanza(3, 2)
salone = Stanza(4, 6)
print(cucina.profondita())
print(bagno.profondita())
print(salone.ampiezza())
print(salone.area())
## print(bagno._profondita)
```

```
5
2
4
24
```

In [ ]:
```
dir(salone)
```

In [31]:
```
salone2 = salone
salone3 = Stanza(4, 6)
```

In [32]:
```
print(salone, salone2, salone3, sep="\n")
```

```
<__main__.Stanza object at 0x700000002524550>
<__main__.Stanza object at 0x700000002524550>
<__main__.Stanza object at 0x700000002524730>
```

In [35]:
```
print(salone2 is salone)
print(salone3 is salone)
```

```
True
False
```

In [36]:
```
print(salone2 == salone)
print(salone3 == salone)
```

```
True
False
```

In [45]:
```
class Stanza:
    def __init__(self, dim1, dim2):
        self._ampiezza   = dim1
        self._profondita = dim2
    def ampiezza(self):
        return self._ampiezza
    def profondita(self):
        return self._profondita
    def area(self):
        return self._ampiezza * self._profondita
    def __repr__(self):
        return f"Stanza di dimensioni {self._ampiezza} x {self._profondita}\

cucina = Stanza(4, 5)
bagno  = Stanza(3, 2)
salone = Stanza(4, 6)
print(cucina)
print(bagno)
print(salone)
print(salone.area())
```

```
Stanza di dimensioni 4 x 5
<__main__.Stanza object at 0x7000000024b3910>
Stanza di dimensioni 3 x 2
<__main__.Stanza object at 0x7000000022a5040>
Stanza di dimensioni 4 x 6
<__main__.Stanza object at 0x700000002507cd0>
24
```

In [41]:
```
nuova_instanza_generica = object()
```

In [42]:
```
print(nuova_instanza_generica)
```

```
<object object at 0x700000002008c60>
```

In [44]:
```
print(dir(nuova_instanza_generica))
```

```
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '
__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subcl
ass__', '__le__', '__lt__', '__ne__', '__new__', '__reduce__', '__reduce_ex_
_', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__']
```

---

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 [46]:
```python
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 super().__repr__() + " insegna nel modulo di " + self._topic

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

In [47]:
```
andrea = Docente("Andrea", "Ribuoli", "Python")
print(andrea)
andrea.setAge(62)
rossana = Persona("Rossana", "Carbone")
print(rossana)
rossana.setAge(25)
eleonora = Student("Eleonora", "Moroni", 1)
print(eleonora)
```

```
Andrea Ribuoli insegna nel modulo di Python
Rossana Carbone
Eleonora Moroni frequenta l'anno 1
```

lattina.py

pagina 601 esercizio **P9.5**

Realizzate una classe **SodaCan** che rappresenti una lattina di bibita (quindi, di forma cilndrica), dotata dei metodi:

- *getSurfaceArea()* e
- *getVolume()*,

che restituiscano, rispettivamente:

- la superficie totale e
- il volume della lattina

Il costruttore riceve, come argomenti, l'altezza e il raggio della base della lattina.

In [56]:
```python
import math
class SodaCan:
    def __init__(self, altezza, raggio):
        self._altezza = altezza
        self._raggio = raggio
    def getSurfaceArea(self):
        circonferenza = 2 * math.pi * self._raggio
        laterale = self._altezza * circonferenza
        area_base = math.pi * self._raggio ** 2
        return laterale + 2 * area_base
    def getVolume(self):
        area_base = math.pi * self._raggio ** 2
        return self._altezza * area_base

mini_lattina = SodaCan(7.6, 2.5)
lattina = SodaCan(9.24, 3.37)

print(mini_lattina.getSurfaceArea())
print(lattina.getSurfaceArea())
print(mini_lattina.getVolume())
print(lattina.getVolume())
```

```
158.65042900628455
267.00835785831157
149.22565104551518
329.6716833337982
```

---

quiz.py

pagina 601 esercizio **P9.7**

Realizzate una classe **Student** che rappresenti uno studente. Uno studente ha un nome, un cognome e un punteggio totale (*score*)

Definite nella classe:

- un costruttore appropriato e i metodi
- *getName()* (restituisce anche il cognome...)
- *addQuiz(score)*
- *getTotalScore()*
- *getAverageScore()*

Per realizzare l'ultimo dei metodi richiesti è necessario memorizzare il numero di questionari compilati.

In [72]:
```python
class Student:
    def __init__(self, nome, cognome):
        self._nome = nome
```

```
        self._cognome = cognome
        self._totScore = 0.0
        self._nOfQuiz = 0
    def getName(self):
        return f"{self._nome} {self._cognome}"
    def addQuiz(self, score):
        self._totScore += score
        self._nOfQuiz += 1
    def getTotalScore(self):
        return self._totScore
    def getAverageScore(self):
        if self._nOfQuiz == 0:
            raise ValueError(f"Lo studente {self._nome} {self._cognome}" +
                             " non ha sostenuto prove")
        return self._totScore / self._nOfQuiz
```

In [74]:
```
eleonora = Student("Eleonora", "Moroni")
davide = Student("Davide", "Sambughi")
```

In [75]:
```
davide.addQuiz(19)  # self._nOfQuiz -->  1
```

In [76]:
```
davide.addQuiz(17)  # self._nOfQuiz -->  2
davide.addQuiz(18)  # self._nOfQuiz -->  3
```

In [77]:
```
davide.getAverageScore()
```

Out[77]: 18.0

In [78]:
```
eleonora.addQuiz(20)
```

In [79]:
```
eleonora.getAverageScore()
```

Out[79]: 20.0

In [80]:
```
edoardo = Student("Edoardo", "Caprini")
```

In [81]:
```
try:
    edoardo.getAverageScore()
except ValueError as e:
    print(e)
```

```
Lo studente Edoardo Caprini non ha sostenuto prove
```

In [57]:
```
!python3 resoconto.py lattina
```

```
 1  Mirco  Azzolini
 2  Wallace  Bezerra Beretta
 3  Alexandru Razvan  Brasovianu
 4  Edoardo  Caprini
 5  Maryuri  Catozzi
 6  Federico  De Grandis
 7  Maikol  Freddari
 8  Sofia  Gaona
 9  Alessia  Gasparini
10  Enrico  Giorgi
11  Andrea  Kanakciu
12  Francesco  Marinelli
13  Filippo  Martino
14  Eleonora  Moroni
15  Norman  Muzi
16  Mattia  Roberti
17  Alessandro  Rovinelli
18  Davide  Sambughi
19  Maximiliano  Serafini
20  Giovanni  Sperandini
21  Alessio  Stomeo
22  Lesly Pierina  Vera Castillejo
```

In [71]:
```
!python3 resoconto.py quiz
```

```
 1  Mirco  Azzolini
 2  Wallace  Bezerra Beretta
 3  Alexandru Razvan  Brasovianu
 4  Edoardo  Caprini
 5  Maryuri  Catozzi
 6  Federico  De Grandis
 7  Maikol  Freddari
 8  Sofia  Gaona
 9  Alessia  Gasparini
10  Enrico  Giorgi
11  Andrea  Kanakciu
12  Francesco  Marinelli
13  Filippo  Martino
14  Eleonora  Moroni
15  Norman  Muzi
16  Mattia  Roberti
17  Alessandro  Rovinelli
18  Davide  Sambughi
19  Maximiliano  Serafini
20  Giovanni  Sperandini
21  Alessio  Stomeo
22  Lesly Pierina  Vera Castillejo
```

---

print() named parameters

- **sep** `" "`

- **end** "\n"

- **file** `stdout`

In [86]:
```
daScrivere = open("report.txt", "w")
print("Ecco come posso scrivere un testo", file=daScrivere)
daScrivere.close()
```

In [87]:
```
daScrivere = open("report.txt", "a")
print("Aggiungo nuovo testo", file=daScrivere)
daScrivere.close()
```

In [90]:
```
daLeggere = open("report.txt", "r")
print(daLeggere.readline(), end="")
print(daLeggere.readline(), end="")
daLeggere.close()
```

```
Ecco come posso scrivere un testo
Aggiungo nuovo testo
```

In [92]:
```
daLeggere = open("report.txt", "r")
riga = daLeggere.readline()
while riga != "":
    print(riga, end="")
    riga = daLeggere.readline()
daLeggere.close()
```

```
Ecco come posso scrivere un testo
Aggiungo nuovo testo
```

In [98]:
```
daLeggere = open("report.txt", "r")
righe = daLeggere.readlines()
for riga in righe:
    print(riga, end="")
daLeggere.close()
```

```
Ecco come posso scrivere un testo
Aggiungo nuovo testo
```

In [99]:
```
daLeggere = open("report.txt", "r")
righe = daLeggere.readlines()
for riga in righe:
    print(riga[0:-1])
daLeggere.close()
```

```
Ecco come posso scrivere un testo
Aggiungo nuovo testo
```

In [100…
```
def leggiFileScelto():
    filename = input("Indica un nome file da leggere: ")
    daLeggere = open(filename, "r")
    righe = daLeggere.readlines()
    for riga in righe:
        print(riga[0:-1])
    daLeggere.close()
```

In [101…
```
leggiFileScelto()
```

```
Indica un nome file da leggere:  report.txt
Ecco come posso scrivere un testo
Aggiungo nuovo testo
```

In [103…
```
leggiFileScelto()
```

```
Indica un nome file da leggere:  report.txt
Ecco come posso scrivere un testo
Aggiungo nuovo testo
Nuova riga di testo non aggiunta tramite Python
```

In [104…
```
def copiaFileScelto():
    filename = input("Indica un nome file da leggere: ")
    filename2 = input("Indica un nome file da scrivere: ")
    daLeggere = open(filename, "r")
    righe = daLeggere.readlines()
    daLeggere.close()
    daScrivere =  open(filename2, "w")
    for riga in righe:
        print(riga[0:-1], file=daScrivere)
    daScrivere.close()
```

In [105…
```
copiaFileScelto()
```

```
Indica un nome file da leggere:  report.txt
Indica un nome file da scrivere:  copia_report.txt
```

In [107…
```
filename2 = input("Indica un nome file da scrivere: ")
daScrivere =  open(filename2, "w")
righe = [f"Riga numero {_:2d}" for _ in range(1, 21)]
for riga in righe:
    print(riga, file=daScrivere)
daScrivere.close()
```

```
Indica un nome file da scrivere:  genero_dati.txt
```

In [96]:
```
type(righe)
```

Out[96]: list

lettera.py

pagine 602-603 esercizio **P9.11**

```
Dear John:

I am sorry we must part.
I wish you all the best.

Sicerely,

Mary
```

In [111…
```
!python3 resoconto.py lettera
```

```
 1  Mirco  Azzolini
 2  Wallace  Bezerra Beretta
 3  Alexandru Razvan  Brasovianu
 4  Edoardo  Caprini
 5  Maryuri  Catozzi
 6  Federico  De Grandis
 7  Maikol  Freddari
 8  Sofia  Gaona
 9  Alessia  Gasparini
10  Enrico  Giorgi
11  Andrea  Kanakciu
12  Francesco  Marinelli
13  Filippo  Martino
14  Eleonora  Moroni
15  Norman  Muzi
16  Mattia  Roberti
17  Alessandro  Rovinelli
18  Davide  Sambughi
19  Maximiliano  Serafini
20  Giovanni  Sperandini
21  Alessio  Stomeo
22  Lesly Pierina  Vera Castillejo
```

In [110…
```
!python3 setallievi.py
```

In [1]:
```
class Letter:
    def __init__(self, letterFrom, letterTo):
        self._letterFrom = letterFrom
        self._letterTo = letterTo
        self._body = []

    def addLine(self, line):
        self._body.append(line)

    def getText(self):
        lines = [f"Dear {self._letterTo}:", ""]
        lines.extend(self._body)
        lines.append("")
        lines.append("Sincerely,")
        lines.append("")
        lines.append(self._letterFrom)
        return "\n".join(lines)
```

In [2]:
```
missiva = Letter("Mary", "John")
missiva.addLine("I am sorry we must part.")
missiva.addLine("I wish you all the best.")
print(missiva.getText())
```

```
Dear John:

I am sorry we must part.
I wish you all the best.

Sincerely,

Mary
```

In [3]:
```
class Letter:
    def __init__(self, letterFrom, letterTo):
        self._letterFrom = letterFrom
        self._letterTo = letterTo
        self._body = []

    def addLine(self, line):
        self._body.append(line)

    def __repr__(self):
        lines = [f"Dear {self._letterTo}:", ""]
        lines.extend(self._body)
        lines.append("")
        lines.append("Sincerely,")
        lines.append("")
        lines.append(self._letterFrom)
        return "\n".join(lines)
```

In [4]:
```
missiva = Letter("Mary", "John")
missiva.addLine("I am sorry we must part.")
missiva.addLine("I wish you all the best.")
print(missiva)
```

```
Dear John:

I am sorry we must part.
I wish you all the best.

Sincerely,

Mary
```

In [5]:
```
missiva2 = missiva
missiva2.addLine("Contenuto aggiunto sulla copia")
print(missiva2)
```

```
Dear John:

I am sorry we must part.
I wish you all the best.
Contenuto aggiunto sulla copia

Sincerely,

Mary
```

In [6]:
```
missiva2 is missiva
```

Out[6]:
```
True
```

In [7]:
```
print(missiva)
```

```
Dear John:

I am sorry we must part.
I wish you all the best.
Contenuto aggiunto sulla copia

Sincerely,

Mary
```

In [9]:
```
lista1 = [_ for _ in range(1, 11)]
```

In [10]:
```
lista2 = list(lista1)
```

In [11]:
```
lista2
```

Out[11]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [12]:
```
lista2 is lista1
```

Out[12]:
```
False
```

In [13]:
```
class Letter:
    def __init__(self, letterFrom=None, letterTo=None):
        if letterFrom == None:
            self._letterFrom = "Unknown"
        else:
            self._letterFrom = letterFrom
        if letterTo == None:
            self._letterTo = "Unknown"
        else:
            self._letterTo = letterTo
        self._body = []

    def addLine(self, line):
        self._body.append(line)

    def __repr__(self):
        lines = [f"Dear {self._letterTo}:", ""]
```

```
        lines.extend(self._body)
        lines.append("")
        lines.append("Sincerely,")
        lines.append("")
        lines.append(self._letterFrom)
        return "\n".join(lines)
```

In [18]:
```
l1 = Letter("John", "Mary")
l2 = Letter("John")
l3 = Letter()
print(l3)
```

```
Dear Unknown:


Sincerely,

Unknown
```

In [19]:
```
# somma(1, 2)      3
# somma(1, 2, 3)   6

def somma(*addendi) :
    print(addendi)
```

In [20]:
```
somma(1, 2)
somma(1, 2, 3)
```

```
(1, 2)
(1, 2, 3)
```

In [21]:
```
def somma(*addendi) :
    s = 0
    for addendo in addendi:
        s += addendo
    return s
```

In [24]:
```
print(somma(1, 2))
print(somma(1, 2, 3))
print(somma(1, 2, 3, 4))
print(somma(1, 2, 3, 4, 5))
```

```
3
6
10
15
```

In [25]:
```
def elabora(**kwargs):
    print(kwargs)
```

In [26]:
```
elabora(sep="-")
```

```
{'sep': '-'}
```

In [27]:
```
elabora(sep="-", end="<>")
```

```
{'sep': '-', 'end': '<>'}
```

In [43]:
```
class Letter:
    def __init__(self, letterFrom=None, letterTo=None):
        if type(letterFrom) == Letter:
            self._letterFrom = letterFrom._letterFrom
            self._letterTo   = letterFrom._letterTo
            self._body       = list(letterFrom._body)
        else:
            if letterFrom == None:
                self._letterFrom = "Unknown"
            else:
                self._letterFrom = letterFrom
            if letterTo == None:
                self._letterTo = "Unknown"
            else:
                self._letterTo = letterTo
            self._body = []

    def addLine(self, line):
        self._body.append(line)

    def __repr__(self):
        lines = [f"Dear {self._letterTo}:", ""]
        lines.extend(self._body)
        lines.append("")
        lines.append("Sincerely,")
        lines.append("")
        lines.append(self._letterFrom)
        return "\n".join(lines)
```

In [44]:
```
missiva = Letter("Mary", "John")
missiva.addLine("I am sorry we must part.")
missiva.addLine("I wish you all the best.")
print(missiva)
```

```
Dear John:

I am sorry we must part.
I wish you all the best.

Sincerely,

Mary
```

In [50]:
```
missiva2 = Letter(missiva)
print(missiva2)
```

```
Dear John:

I am sorry we must part.
I wish you all the best.

Sincerely,

Mary
```

In [51]:
```
missiva2.addLine("Nuova riga")
```

In [52]:
```
print(missiva)
print(missiva2)
```

```
Dear John:

I am sorry we must part.
I wish you all the best.

Sincerely,

Mary
Dear John:

I am sorry we must part.
I wish you all the best.
Nuova riga

Sincerely,

Mary
```

In [57]:
```
print(list(set([1,2,3,2])))
```

```
[1, 2, 3]
```

In [59]:
```
print(list("abc"))
```

```
['a', 'b', 'c']
```