Introduction to Python programming

Course and exam presentation, notebooks and first steps

- Armando Camerlingo

IR0000032 – ITINERIS, Italian Integrated Environmental Research Infrastructures System
(D.D. n. 130/2022 - CUP B53C22002150006) Funded by EU - Next Generation EU PNRR-Mission 4 “Education and Research” - Component 2: “From research to business” - Investment 3.1: “Fund for the realisation of an integrated system of research and innovation infrastructures”

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani

Introduzione alla Programmazione in Python

Armando Camerlingo

03/06/2025

- Armando Camerlingo
- PhD @ Unitus
- HPC/AI Specialist @ Prisma S.p.A

ITINERIS

PRISMA

- Indirizzo mail: armando.camerlingo@unitus.it

Cosa è Python?

- Un linguaggio di programmazione ad alto livello;
- Orientato ad oggetti;
- Facile da leggere e quindi da imparare;
- Supporto di molte librerie per sviluppo in vari ambiti;
- Uso dell’indentazione nella struttura del codice;

Google CoLab

Variabili

- Usate per contenere informazioni;
- A differenza di altri linguaggi, in Python non c'è bisogno di dichiarare le variabili precedentemente all'utilizzo;
- Una variabile è semplicemente un'etichetta che si riferisce ad un oggetto;
- Attenzione: Python è case-sensitive, la variabile x è diversa dalla variabile X!;
- Per adesso ci concentriamo su due tipi di variabili:
    - Stringhe, ovvero insiemi di caratteri;
    - Dati di tipo numerico;

Le variabili si assegnano con l'operatore "="

```
message = "Hello Python World!"
print(message)

n = 30
print(n)
```

Essendo solo etichette, si può riutilizzare la stessa variabile per diversi oggetti, addirittura di diverso tipo:

```
message = “Hello Python World!”
print(message)

message = “Python è il mio linguaggio
preferito!”
print(message)

message = 40
print(message)
```

- Tipi di dati numerici: interi e floating-point;
- Si può controllare il tipo delle variabili con la funzione `type()`:

```
message = “Hello Python World!”
type(message)

number = 30
type(number)

pi = 3.1415
type(pi)
```

Operatori:

Operatore	Operazione
+	Addizione
-	Sottrazione
*	Moltiplicazione
/	Divisione
**	Potenza
%	Modulo

Ordine delle operazioni: PEMDAS

- **P**arentesi: Precedenza più alta, importante utilizzarle correttamente per definire l'ordine in cui eseguire le operazioni;
- **E**xponentiation (Potenze);
- **M**oltiplicazioni e **D**ivisioni;
- **A**ddizioni e **S**ottrazioni;

```
print("Parentesi")
a = 2 * (3-1)
b = (1+1)**(5-2)
c = 1+1**5-2
print(a)
print(b)
print(c)
print("Potenza")
c = 2**2+1
d = 3*1**3
print(c)
print(d)
```

```
print("Moltiplicazione | Divisione >
Addizione | Sottrazione")
e = 2*3-1
f = 6+4/2
g = 4 / 2 * 3
h = 6/3
print(type(h))
print(e)
print(f)
print(g)
```

Stampare Stringhe e Numeri

- Le variabili numeriche possono essere stampate utilizzando il comando `print();`
- Se si vogliono stampare variabili numeriche come parte di una stringa, bisogna usare la funzione `str();`
- Una volta trasformate in stringhe, si possono applicare gli operatori come visto precedentemente;

```
x = 4
print(x)
y = 3*x
print(”x = ” + str(x))
print(”x = ” + str(y))
```

```
a = 8
b = 3
sa = str(a)
sb = str(b)

print(type(a))
print(type(sa))

print(a+b)
print(sa+sb)
```

Si possono unire variabili e stringhe nel print():

```
Nome = “Gianni”
Eta = 30

print(Nome, “ ha “, Eta, “ anni.”)
```

Variabili Booleane e operatori di confronto

ITINERIS

- Una variabile o espressione booleana è un oggetto che può essere solo **`True`** (vera) o **`False`** (falsa);
- `True` e `False` sono valori speciali di tipo `bool`, non stringhe!
- `True` e `False` sono anche keywords di Python, non possono essere usate per altri scopi;
- Una variabile booleana si ottiene ogni volta che si effettua una operazione di confronto tra due numeri, due stringhe o due bool;

Operatore	Confronto
==	sono uguali?
!=	sono diversi?
>	è maggiore?
>=	è maggiore o uguale?
<	è minore?
<=	è minore o uguale?

```
print(1 == 1)
print(1 == 2)
print("a" == "a")
print("a" == "b")

print(type(True))
print(type(1*2 == 2))
```

- 2 oggetti sono uguali se hanno lo stesso valore;
- Si può fare un confronto di uguaglianza tra due numeri, due stringhe e anche altri oggetti;
- Anche in altri linguaggi, due segni di uguale indicano l’operatore di uguaglianza;
- Attenzione, un solo uguale **=** assegna un valore alla variabile, **==** è un operatore relazionale;

```
print(1 == 1)
print(1 == 1.0)
print(1 == '1')

print("stringa" == 'stringa')
print("stringa" == 'Stringa')
print("stringa" == 'stringa ')
```

- Due oggetti sono diversi se non hanno lo stesso valore;
- In Python, per testare la disuguaglianza si usa il punto interrogativo seguito dal segno di uguale;
- A volte ha senso utilizzare l'operatore di disuguaglianza al posto di quello di uguaglianza, la maggior parta della volte sono interscambiabili;

```
print(1 != 0)
print(5 != 5)
```

- Altri operatori di confronto sono i segni di maggiore, minore, maggiore o uguale, minore o uguale:

```
print(1 > 0)
print(1 >= 1)
print(1 < 0)
print(1 <= 1)
```

Liste:

- Una lista è una sequenza di valori (come le stringhe sono sequenze di caratteri);
- In una lista possono esserci variabili di vario tipo;
- Componenti di una lista sono chiamati elementi o oggetti (items);
- Una lista può essere salvata in una variabile;
- Si possono creare in vari modi, il più semplice è con [];

```
studenti = ['Aldo', 'Bruna', 'Carlo']
numeri = [10, 20, 30, 40]
lista_mista = [1, 'pippo', 2.3]
```

Un altro modo di creare e maneggiare le liste è con i metodi che fornisce Python:

Metodo	Operazione
append()	Aggiungi elemento alla fine della lista
pop()	Rimuovi elemento alla fine della lista
insert()	Aggiungi elemento nella lista alla posizione scelta
del list[]	Rimuovi elemento nella lista alla posizione scelta
remove()	Rimuovi elemento con il valore scelto
sort()	Ordina la lista
reverse()	Inverti l'ordine della lista
len()	Ottieni lunghezza della lista

```
lista = []
print('Aggiungiamo elementi alla lista
con append')

lista.append('Carlo')
lista.append('Bruno')

print('Aggiungiamo elementi alla lista
con insert')

lista.insert(1, 'Aldo')
print(lista)
```

```
print(lista)
print("Cancelliamo un elemento con del")
del lista[1]
print(lista)
```

```
print(lista)
print("Cancelliamo un elemento con pop")
lista.pop()
print(lista)
```

```
print(lista)
print("Cancelliamo un elemento con remove")
lista.remove('Carlo')
print(lista)
```

```
print(lista)
print('Ordiniamo la lista con sort')
lista.sort()
print(lista)
```

```
print(lista)
print('Ordiniamo la lista con sort con ordine
invertito')
lista.sort(reverse=True)
print(lista)
```

```
print(lista)
print('Invertiamo la lista con reverse')
lista.reverse()
print(lista)
```

```
print(lista)
print('Lunghezza della lista')
len(lista)
```

Accedere agli elementi della lista

Gli elementi della lista sono identificati dal loro valore e dalla loro posizione nella lista, che inizia da **0** :

```
print(lista[0])
print(lista[1])
```

Non sempre si sa la lunghezza della lista, quindi per accedere agli ultimi elementi si possono usare gli indici negativi. L'ultimo elemento avrà indice **-1**, e così via:

```
print(lista[-1])
print(lista[-2])
```

Modificare gli elementi della lista

Sapendo la posizione dell'elemento, si può modificare:

```
lista[-1] = 'Dario'
print(lista)
```

Slicing delle liste

L'operatore di slice [n:m] ritorna una lista dal n-esimo elemento al (m-1)-esimo elemento, ovvero compreso l'elemento n ed escluso l'elemento m. Se l'indice non è specificato, si va dall'inizio o dalla fine della lista.

```
alf = ['a','b','c','d','e','f']
print(alf[1:3])
print(alf[:3])
print(alf[:])
print(alf[-3:-1])
```

Funzioni per liste numeriche

Ci sono delle funzioni utili per maneggiare le liste contenenti numeri:

Funzione	Operazione
range(), seguito da list()	generare liste di numeri
sum()	Restituisce la somma dei numeri nella lista
min()	Restituisce il minimo della lista
max()	Restituisce il massimo della lista

- `range()` è una funzione implementata in Python che restituisce un oggetto di tipo range, ovvero una sequenza di numeri interi. Si possono dare 3 argomenti di input: l'inizio della sequenza, la fine della sequenza e l'intervallo tra i due numeri consecutivi (step).
- `list()` trasforma altri oggetti compatibili in liste.

```
seq = range(1,100)
print(seq)
print(type(seq))
seq_2 = range(1,100,2)
print(seq_2)
```

```
lista = list(seq)
print(type(list))
print(lista)
lista_2 = list(seq_2)
print(type(lista_2))
print(lista_2)
```

- max() restituisce il massimo della lista;
- min() restituisce il minimo della lista;
- sum() restituisce la somma degli elementi della lista;

```
numeri = [14, -25, 4, 56, -13, 0]
max = max(numeri)
min = min(numeri)
somma = sum(numeri)

print('massimo = ', max)
print('minimo = ', min)
print('somma = ', somma)
```

- Con l’operatore in si può controllare se un item è all’interno di una lista:

```
lettere = ['a', 'b', 'c', 'd', 'e']
print('a' in lettere)
print('f' in lettere)
```

Operatori Logici

- Questi operatori confrontano condizioni e restituiscono un valore bool;
- Ci sono 3 operatori logici: `and` (e), `or` (o) e `not` (non);
- il significato è facilmente capibile dal nome degli operatori;

	Condizione _1	Condizione _2	Output
and	True	True	True
	True	False	False
	False	True	False
	False	False	False
or	True	True	True
	True	False	True
	False	True	True
	False	False	False

	Condizione	Output
not	True	False
	False	True

```
print(1>0 and "a" == "a")
print(1>0 and 0 > 8)

print(1>0 or 0 > 8)
print(1<0 or 0 > 8)

print(not 1==0)
```

IF ELSE

- Nei programmi è utile avere la possibilità di eseguire insiemi di istruzioni solo se una certa condizione è soddisfatta. Le istruzioni condizionali servono a questo scopo;

```
if x > 0:
    print("x è positivo")
```

- L'espressione booleana dopo `if` è chiamata condizione. Se è `True`, il codice indentato viene eseguito, se è `False` non succede niente;

- I blocchi `if` hanno la stessa struttura delle funzioni: un *header* seguito da un *body* indentato. Non ci sono limiti a quante istruzioni possono essere nel body, ma deve essercene almeno una;

```
a = 10
if a > 0:
  print("a è maggiore di 0")

a = 1000
b = 50
if b > a:
  print("b è maggiore di a")

studenti = ['Aldo', 'Bruno', 'Carlo']
if len(studenti) > 5:
  print("La classe è affollata")
```

- Una seconda forma del blocco if è data dall'aggiunta di un'alternativa, ovvero dare due possibilità e la condizione determina quale delle due viene eseguita:

```
If condizione is True:
     # esegui questa istruzione
else:
     # esegui quest'altra condizione
```

```
x = 24
if x%2 == 0:
  print(x, "è un numero pari")
else:
  print(x, "è un numero dispari")
```

- Molte volte, si vuole provare una serie di condizioni invece di una sola;
- Si può fare con `if-elif-else`;
- Non c'è limite al numero di condizioni;
- Sempre necessario uno statement `if` e non più di un `else`, nessun limite sugli `elif`;

```
a = 0
b = 1

if a > b:
  print("a è maggiore di b")
elif a < b:
  print("a è minore di b")
else:
  print("a è uguale a b")
```

Esercizio

Scrivere uno snippet di codice per verificare se un anno è bisestile, ovvero un anno divisibile per 4 ma non per 100, oppure divisibile per 400.

Esempio: 1700, 1800, 1900 non sono anni bisestili, mentre 1600 e 2000 sì.

Soluzione

```
anno = 1871

if (anno%400 == 0) or (anno%4 == 0 and
anno%100 != 0):
  print(anno, "è un anno bisestile")
else:
  print(anno, "non è un anno bisestile")
```

Cicli Iterativi (Loops)

- Il programmi sono usati spesso per compiere compiti ripetitivi. Quindi è comune avere la necessità di ripetere in modo identico una serie di operazioni.
- I cicli iterativi sono strutture che compiono diverse iterazioni di un blocco di codice. Un'**iterazione** è l'esecuzione ripetuta di una stessa serie di istruzioni. Ci sono due tipi di cicli:
  - Cicli ad iterazioni definite: il numero di ripetizioni è determinato e specificato in anticipo (istruzione **`for`**);
  - Cicli ad iterazioni indefinite: il codice viene ripetuto finchè non è soddisfatta una condizione (istruzione **`while`**);

Ciclo FOR

Un ciclo FOR ha la seguente struttura:

```
for <var> in <iterabile>:
    <codice>
```

- `<iterabile>` è una collezione di oggetti - per esempio una lista;
- Il `<codice>` nel corpo del ciclo sono individuati dall'indentazione e sono eseguiti una volta per ogni elemento dell' `<iterabile>`;
- la variabile del ciclo `<var>` assume il valore di ogni elemento in `<iterabile>` in successione per ogni iterazione;

```
for n in [0, 1, 2 , 3]:
  print(n)
print('finito il ciclo')
```

Possiamo usare la funzione range per crearci liste numeriche di grandezza desiderata per scrivere il ciclo for:

```
for i in range(1,10):
    print(i)
```

- Si possono usare anche liste di oggetti diversi da numeri:

```
lista = ['Aldo', 'Bruno', 'Carlo', 'Dario']

for nome in lista:
  print(nome)
```

Esercizio

Stampare tutti i numeri da 1 a 50 e per ogni numero stampare se è un multiplo di 5 o no

Soluzione

```
numeri = range(1,51)

for num in numeri:
  print("numero ", num)
  if num%5 == 0:
    print(num, "è un multiplo di 5")
  else:
    print(num, "non è un multiplo di 5")
```

Esercizio

Stampare tutti i numeri da 1 a 100 e per ogni numero stampare se è un multiplo di 12 o no. Creare anche un contatore che restituisce quanti multipli si sono trovati.

Soluzione

```
numeri = range(1,101)
contatore = 0

for num in numeri:
  if num%12 == 0:
    print(num, "è un multiplo di 12")
    contatore = contatore + 1
    #contatore += 1

print("Sono stati trovati", contatore, "multipli
di 12")
```

- Quando si cicla usando una lista, può essere utile sapere conosce l'indice (la posizione nella lista) dell'oggetto considerato;
- Si può usare il metodo `list.index(value)`, ma c'è un modo più semplice;
- La funzione `enumerate()` prende in input una lista e restituisce due oggetti contenenti rispettivamente gli indici e gli elementi della lista

```
parola = 'Python'

print("ecco lo spelling di ", parola)
for idx, lettera in enumerate(parola):
  print(lettera)
```

Esercizio

Contare quante volte la lettera e compare nel primo paragrafo della pagina wikipedia inglese di Python (fate copia e incolla)

Soluzione

```
testo = '<inserire testo>'

contatore = 0
for idx, carattere in enumerate(testo):
    if carattere == 'e':
      contatore += 1

print("Ci sono ", contatore, "e")
```

Ciclo WHILE

ITINERIS

- Il ciclo `While` ha la seguente struttura:

```
while <condizione>:
    <codice>
```

- Il `<codice>`, individuato dall'indentazionesono le istruzioni da essere ripetute;
- l'iterazione viene eseguita solo la `<condizione>` è vera, ovvero ha valore `True;`

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

Esercizio

Usare un ciclo while per calcolare il fattoriale di un numero intero, ovvero n! = n*(n-1)*(n-2)*...*1

Soluzione

```
numero = 8
tmp = numero
fatt = 1
while tmp > 1:
   fatt = fatt * (tmp-1)
   tmp = tmp - 1

print( numero, " ! = ", fatt )
```

THANKS!

IR0000032 – ITINERIS, Italian Integrated Environmental Research Infrastructures System
(D.D. n. 130/2022 - CUP B53C22002150006) Funded by EU - Next Generation EU PNRR-Mission 4 “Education and Research” - Component 2: “From research to business” - Investment 3.1: “Fund for the realisation of an integrated system of research and innovation infrastructures”

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani