Nel seguito delle note saranno presenti codici in dei riquadri e, per completezza, dopo la riga [Output] viene presentato anche il risultato degli stessi nel caso ci fossero (i.e. ciò che viene stampato su shell).

1 Prima lezione

1.1 Funzione print

Se un macchina fosse senziente e gentile, quini non un'intelligenza artificiale cresciuta su twitter, forse come prima cosa saluterebbe tutti e il modo per comunicare è la funzione print, vediamo quindi il più classico degli esempi:

```
print('Hello world!')

[Output]
Hello world!
```

Questa funzione può stampare sia valori che espressioni:

```
print('Hello world!')
print('42')

print('Hello world!', 42)
print('Adesso vado \n a capo')

[Output]
Hello world!
42
Hello world! 42
Adesso vado
 a capo
```

1.2 Commenti

Al fine di rendere fruibile ad altri o anche al voi stesso del futuro il codice è opportuno inserire i commenti, ovvero frasi che non vengono lette dall'interprete (o dal compilatore) che spiegano cosa voi stiate facendo; altrimenti vi ritroverete nella scomoda situazione in cui solo dio saprebbe spiegarvi il funzionamento del vostro codice.

```
#per i commenti che occupano una singola linea di codice si una il cancelletto
#stampo hello world
print('Hello world!')

"""
per un commento di maggiori linee di codice

vanno usate tre virgole per racchiuderlo
"""
'''
ma van bene

anche tre apici
'''
[Output]
Hello world!
```

1.3 Variabili

Una variabile è un nome, un simbolo, che si da ad un certo valore. In Python non è necessario né definire le variabili prima di utilizzarle, né specificare il loro tipo, esse si creano usando il comando di assegnazione '='. Facciamo un esempio con variabile numeriche:

```
numerointero= 13
numeroavirgolamobile= 13.

print('Numero intero:', numerointero, 'Numero in virgola mobile:', numeroavirgolamobile)
#oppure possiamo stampare in questo modo:
print(f'Numero intero: {numerointero}, Numero in virgola mobile: {numeroavirgolamobile}')

[Output]
Numero intero: 13 Numero in virgola mobile: 13.0
Numero intero: 13 Numero in virgola mobile: 13.0
```

Ovviamente le variabili possono essere non solo numeri ma anche molto altro, e possiamo verificarne il tipo grazie alla funzione 'type()':

```
#inizializziamo delle variabili
n = 7
x = 7.
stringa = 'kebab'
lista = [1, 2., 'cane']
tupla = (42, 'balena')
dizionario = {'calza': 0, 'stampante': 0.5}

#stampiamole e stampiamone il tipo
print(n, type(n))
print(x, type(x))
print(stringa, type(stringa))
print(lista, type(lista))
print(tupla, type(tupla))
print(dizionario, type(dizionario))

[Output]
7 <class 'int'>
7.0 <class 'float'>
kebab <class 'str'>
[1, 2.0, 'cane'] <class 'list'>
(42, 'balena') <class 'tuple'>
{'calza': 0, 'stampante': 0.5} <class 'dict'>
```

Vediamo ora come fare le classiche operazioni matematiche:

```
x1 = 3
y1 = 4

somma = x1 + y1
prodotto = x1*y1
differenza = x1 - y1
rapporto = x1/y1
potenza = x1**y1

print(somma, prodotto, differenza, rapporto, potenza)
[Output]
7 12 -1 0.75 81
```

Vale la pena dilungarsi un attimo su una piccola questione: l'aritmetica in virgola mobile è intrinsecamente sbagliata poiché giustamente il computer ha uno spazio di memoria finita e quindi non può tenere infinite cifre decimali:

```
x = 0.1
y = 0.2
z = 0.3

print(x + y)
print(z)

[Output]
0.30000000000000004
0.3
```

Il precedente è solo uno tra i molti esempi che si potrebbero fare per far notare come l'aritmetica dei numeri in virgola mobile possa dare problemi; Il lettore provi a vedere se è vero che $a + (b + c) = (a + b) + c$ con a, b, c numeri in virgola mobile, probabilmente il computer non sarà d'accordo. Ai computer i numeri in virgola mobile, i numeri reali, non piacciono molto, preferiscono i numeri interi e quelli razionali (e ovviamente adorano le potenze di due, grazie codice binario):

```
#variabili
x = 0.1
y = 0.2
z = 0.3
#sommo le prime due
t = x + y

"""
applico una funzione che mi fornisce
una tupla contenente due numeri interi
il cui rapporto restituisce il numero iniziale.
output del tipo: (numeratore, denominatore)
"""
print(t.as_integer_ratio())
```

```
print(z.as_integer_ratio())

[Output]
(1351079888211149, 4503599627370496)
(5404319552844595, 18014398509481984)
```

Per quanto riguarda i numeri in virgola mobile, possiamo scegliere quante cifre dopo la virgola stampare, vediamolo con un esempio:

```
#definiamo una variabile
c = 3.141592653589793

#stampa come intero
print('%d' %c)

#stampa come reale
print('%f' %c) #di default stampa solo prime 6 cifre
print(f'{c}')  #di default stampa tutte le cifre

#per scegliere il numero di cifre, ad esempio sette cifre
print('%.7f' %c)
print(f'{c:.7f}')

[Output]
3
3.141593
3.141592653589793
3.1415927
3.1415927
```

Notare che il computer esegue un arrotondamento.
Una variabile può essere ridefinita e cambiare valore, il computer userà l'assegnazione più recente:

```
#definiamo una variabile
x = 30

"""

operazioni varie


"""

#ridefiniamo la variabile
x = 18

print('x=', x)

"""
E' possibile anche sovrascrivere una variabile
con un numero che dipende dal suo valore precedente:
"""
x = x + 1 #incrementiamo di uno
#Oppure:
x += 1
print('x=', x)

x = x * 2 #moltiplichiamo per due
#Oppure:
x *= 2
print('x=', x)

[Output]
x= 18
x= 20
x= 80
```

Come si vede i vari comandi $x = x$ *operazione numero* possono essere abbreviati con x *operazione*$=$ *numero*.

1.4 Librerie

Le librerie sono luoghi mistici create dagli sviluppatori, esse contengono molte funzioni, costanti e strutture dati predefinite; in generale se volete fare qualcosa esisterà una libreria con una funzione che implementa quel qualcosa o che comunque vi può aiutare in maniera non indifferente. Prima di poter accedere ai contenuti di una libreria, è necessario importarla. Per farlo, si usa il comando import. Solitamente è buona abitudine importare tutte le librerie che servono all'inizio del file. Ecco un paio di esempi:

```
import numpy

#per usare un contenuto di questa libreria basta scrivere numpy.contenuto
pigreco = numpy.pi
print(pigreco)

#Possiamo anche ribattezzare le librerie in questo modo
import numpy as np
#da ora all'interno del codice numpy si chiama np

eulero = np.e
print(eulero)

[Output]
3.141592653589793
2.718281828459045
```

```
import math

coseno=math.cos(0)
seno = math.sin(np.pi/2) #python usa di default gli angoli in radianti!!!
senosbagliato = math.sin(90)

print('Coseno di 0=', coseno, "\nSeno di pi/2=", seno, "\nSeno di 90=", senosbagliato)

#bisogna quindi stare attenti ad avere tutti gli angoli in radianti
angoloingradi = 45
#questa funzione converte gli angoli da gradi a radianti
angoloinradianti = math.radians(angoloingradi)

print("Angolo in gradi:", angoloingradi, "Angolo in radianti:", angoloinradianti)

[Output]
Coseno di 0= 1.0
Seno di pi/2= 1.0
Seno di 90= 0.8939966636005579
Angolo in gradi: 45 Angolo in radianti: 0.7853981633974483
```

Le due librerie qui usate contengono funzioni simili, ad esempio il seno è implementato sia in numpy che in math, cambia il fatto che math può calcolare il seno di un solo valore, mentre numpy, come vedremo, può calcolare il seno di una sequenza di elementi. Come sapere tutte le possibili funzioni contenute nelle librerie e come usarle? "Leggetevi il cazzo di manga" (n.d.r. Leggetevi la documentazione disponibile tranquillamente online)

1.5 Gestione errori

È molto facile scrivere codice che produca errore, magari perchè distrattamente ci siamo dimenticati qualcosa o magri qualcosa è stato implementato male. Esiste un costrutto che ci permette di gestire gli errori in maniera tranquilla diciamo. Facciamo un semplice esempio:

```
a = 0
b = 1/a
print(b)

[Output]
Traceback (most recent call last):
  File "<tmp 1>", line 5, in <module>
    b = 1/a
ZeroDivisionError: division by zero
```

Abbiamo fatto una cosa molto brutta, nemmeno Dio può dividere per zero (cosa non proprio vera infatti si hanno ancora teorie fisiche con divergenze purtroppo) e quindi il computer ci da errore. Possiamo aggirare il problema, evitando così il second impact, in due modi diciamo:

1.5.1 Try e except

Possiamo utilizzare il costrutto try except dicendo al computer: prova a fare la divisione e, se questa da errore, e l'errore è "ZeroDivisionError" allora assegna a b un altro valore. in questo modo eventuali istruzioni presenti dopo vengono eseguite e il codice non si arresta.

```
a = 0
try :
    b = 1/a
except ZeroDivisionError:
    b = 1
```

```
6
7 print(b)
8
9 [Output]
10 1
```

1.5.2 Raise Exception

Mettiamo il caso in cui ci siano operazioni da fare in cui il valore della variabile "b" è importante, quindi sarebbe meglio interrompere il flusso del codice perché con un dato valore il risultato finale sarebbe poco sensato. Si può fare il controllo del valore e sollevare un'eccezione per fermare il codice.

```
1 """
2 leggo un valore da shell
3 uso del comando try per evitare che venga letto
4 qualcosa che non sia un numero: e.g. una stringa
5 """
6 try:
7     b = int(input('scegliere un valore:'))
8 except ValueError:
9     print('hai digitato qualcosa diverso da un numero, per favore ridigitare')
10     b = int(input('scegliere un valore:'))
11
12 #se si sbaglia a digitare di nuovo il codice si arresta per ValueError
13
14 #controllo se e' possibile proseguire
15 if b > 7 :
16     #se vero si blocca il codice sollevando l'eccezione
17     messaggio_di_errore = 'il valore scelto risulta insensato in quanto nulla supera 7, misura
       massima di ogni cosa'
18     raise Exception(messaggio_di_errore)
19
20 [Output]
21 8
22 Traceback (most recent call last):
23   File "<tmp 1>", line 18, in <module>
24     raise Exception(messaggio_di_errore)
25 Exception: il valore scelto risulta insensato in quanto nulla supera 7, misura massima di ogni
       cosa
```