**INTRODUZIONE**

Un modulo è un raccoglitore esterno di variabili, metodi e funzioni. Per poter utilizzare in python uno di questi oggetti esterni, è necessario importare il modulo. L’insieme delle variabili e delle funzioni è definito entity.

Il modulo si importa in python tramite la keyword import.

Ci sono tantissimi moduli in python. Una lista completa di tali moduli è presente al seguente indirizzo:

<https://docs.python.org/3/py-modindex.html>

Ovviamente i moduli sono tantissimi. Ai nostri fini interessa capirne il funzionamento. Utilizzeremo quindi i moduli più noti in python, ed anche i più semplici. Vedremo, inoltre, come creare nuovi moduli.

**IMPORTARE I MODULI**

Come detto, per importare un modulo si utilizza la keyword import. Per vedere tutti i metodi e le variabili presenti nel modulo si applica ad esso la funzione dir e la si stampa.

Codice Python:

# Importazione modulo math e modulo sys e stampa variabili e metodi:

import math

print("Variabili e metodi del modulo math: ")

print(dir(math))

print()

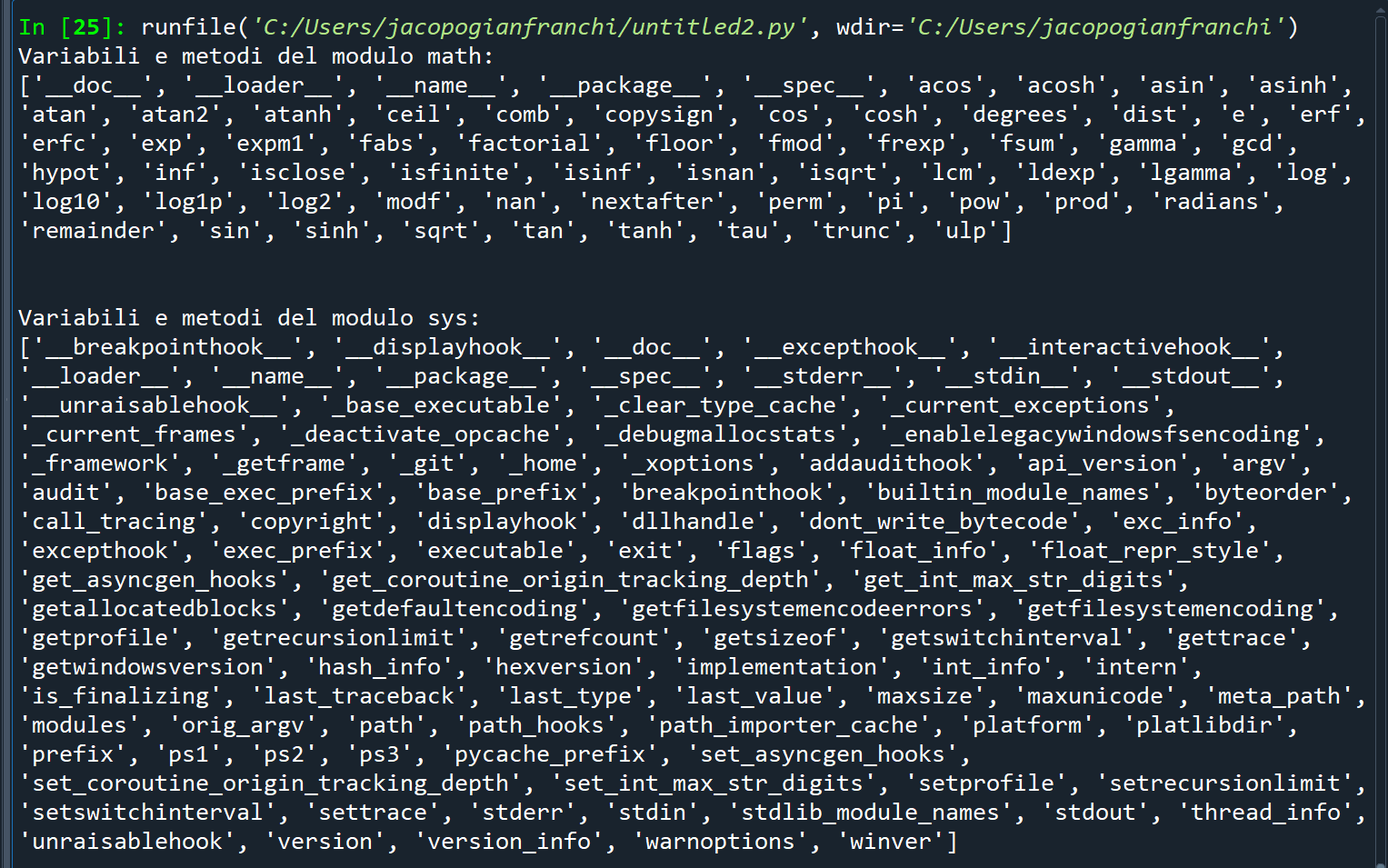
print()

import sys

print("Variabili e metodi del modulo sys: ")

print(dir(sys))

Console:



E’ possibile anche importare due o più moduli contemporaneamente:

Codice Python:

# Importazione contemporanea dei due moduli:

import math, sys

Vediamo un esempio di utilizzo di un modulo.

Supponiamo di voler utilizzare la funzione ceil, funzione presente nel modulo math. Cosa fa questa funzione?

Arrotonda un numero all’intero più vicino. Ad esempio, se passiamo 10.3 come input alla funzione math.ceil, otterremo 11 come output.

Vediamo un esempio.

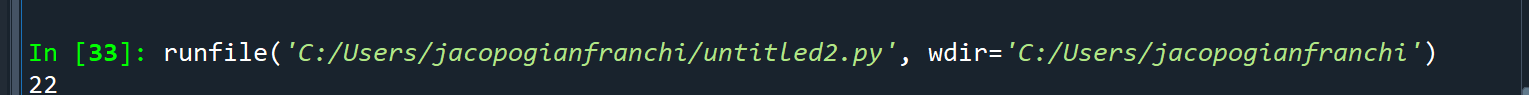
Codice Python:

import math

numero = 21.03

print(math.ceil(numero))

Console:



Nel nostro esempio abbiamo utilizzato solo la funzione ceil. Nella realtà possiamo dover utilizzare più funzioni insieme. Questo fa si che conviene utilizzare un alias al nome del modulo importato:

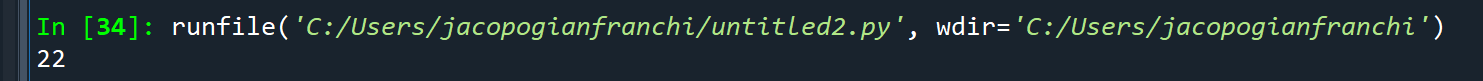
Codice Python:

import math as m

numero = 21.03

print(m.ceil(numero))

Console:



Si noti che è obbligatorio scrivere il nome del modulo, o un suo alias, prima di utilizzare il nome della funzione che si trova all’interno di esso e che ci serve. Infatti, nei nostri esempi, abbiamo utilizzato m.ceil per utilizzare la funzione ceil presente nel modulo m.

Questo perché, nel nostro script, possiamo creare una funzione ceil diversa da quella presente nel modulo math. Vediamo un paio di esempi.

Codice Python:

import math as m

def ceil(numero):

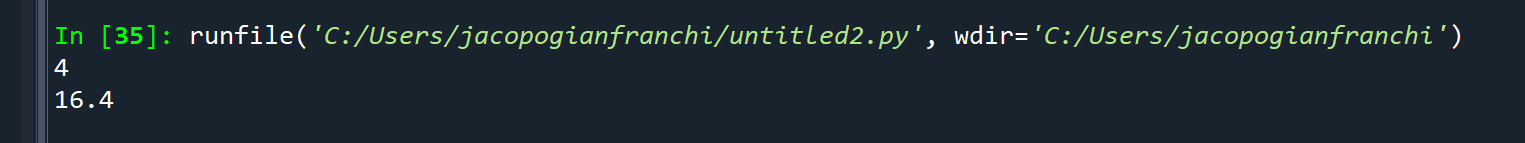
    return numero \* 5

numero = 3.28

print(m.ceil(numero))

print(ceil(numero))

Console:



Per prima cosa abbiamo richiamato la funzione ceil presente nel modulo math: essa ha ritornato 4, ossia il primo numero intero maggiore del numero inserito.

Dopodiché abbiamo richiamato la funzione creata all’interno del nostro script: essa ritorna il numero passato in input moltiplicato per 5.

Si noti che è possibile “scaricare in locale” la funzione interna al modulo desiderato.

E’ tuttavia sconsigliato farlo, in quanto c’è il rischio di trovare all’interno del nostro script una funzione avente lo stesso nome.

Vediamo come scaricare la nostra funzione ceil:

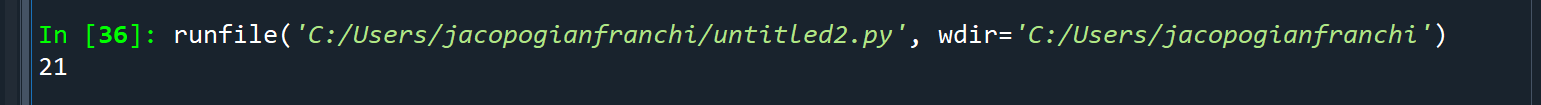
Codice Python:

from math import ceil

numero = 20.21

print(ceil(numero))

Console:



OK

Cosa succede se abbiamo una funzione ceil presente anche nello script?

Dipende dalla posizione del from…import.

Se il from…import si trova prima della definizione della funzione ceil all’interno dello script, verrà eseguita la funzione definita nello script. Se invece il from… import si trova dopo la definizione della funzione nello script, verrà eseguito il ceil “scaricato”.

Vediamo un esempio.

Download della funzione ceil presente in math prima della dichiarazione della funzione ceil nello script:

Codice Python:

from math import ceil

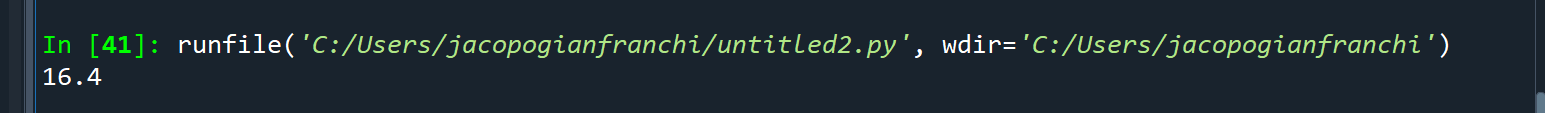
def ceil(numero):

    return numero \* 5

numero = 3.28

print(ceil(numero))

Console:

****

E’ stata quindi eseguita la funzione ceil definita nello script.

Esempio2: Download della funzione ceil presente in math dopo la dichiarazione della funzione ceil nello script:

Codice Python:

def ceil(numero):

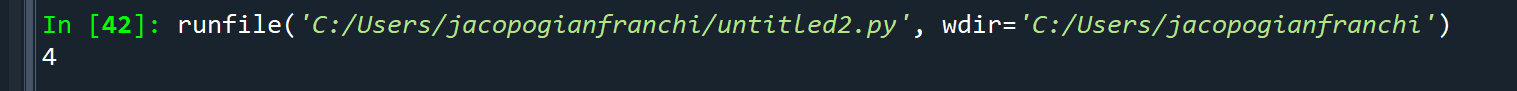
    return numero \* 5

from math import ceil

numero = 3.28

print(ceil(numero))

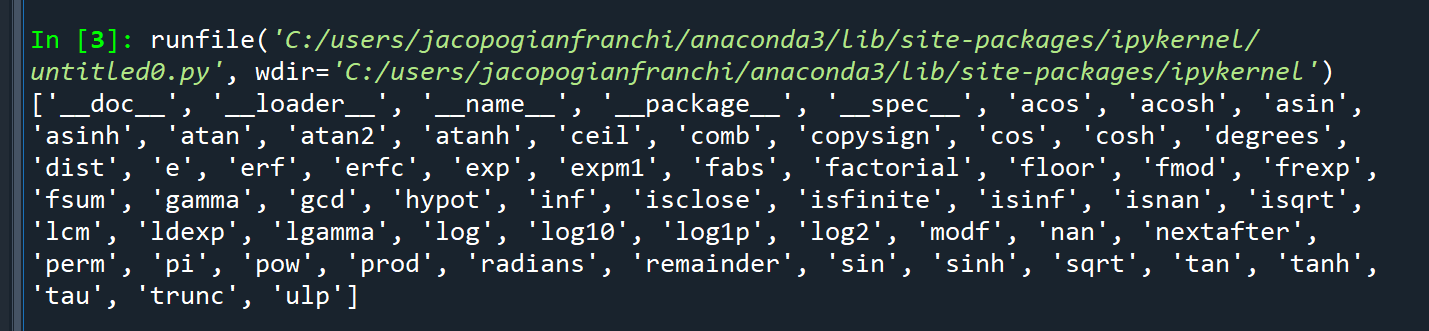
Console:



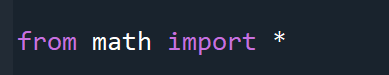
E’ stata quindi utilizzata la funzione ceil scaricata dal modulo math.

E’ ovviamente possibile scaricare tutte le funzioni che si trovano all’interno di un modulo. Come nel caso precedente, anche in questo caso è sconsigliato agire in questo modo, sempre per lo stesso motivo di rischio di sovrapposizione dei nomi delle funzioni.

Finora abbiamo visto la funzione ceil all’interno del modulo math. Possiamo scaricare, giusto a titolo di esempio, anche altre funzioni. Riportiamo qui per comodità tutte le funzioni del modulo math, estratte tramite la funzione dir:



Per scaricarle tutte si utilizza la seguente notazione:



Così abbiamo scaricato tutte le funzioni. Possiamo così utilizzarne varie presenti nel modulo math e ora scaricate in locale. Per comodità, utilizziamo le seguenti funzioni:

factorial

floor

log2

Non ci interessa per ora sapere cosa fanno queste funzioni nel dettaglio: esse sono state considerate al solo scopo di mostrare che è possibile scaricare tutte le funzioni ed è possibile utilizzarne contemporaneamente più di una

Codice Python:

from math import \*

numero = 6

print(factorial(numero))

print(floor(numero))

print(log2(numero))

Console:

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

Ovviamente, come ben sappiamo, non conviene importare tutte le funzioni. Conviene utilizzarle da dentro il modulo:

Codice Python:

import math as m

numero = 6

print(m.factorial(numero))

print(m.floor(numero))

print(m.log2(numero))

Console:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Approfondiremo ora tre moduli molto importanti:

1. Math
2. Random
3. Platform

**MATH MODULE**

Riportiamo qui per comodità le funzioni presenti nel modulo math:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Ai fini dell’esame è necessario conoscere solo sei funzioni:

1. ceil()
2. floor()
3. trunc()
4. factorial()
5. sqrt()
6. hypot()

Vediamo con un singolo script python qual è la funzione di questi metodi:

Codice Python:

print()

import math as m

print("Funzione ceil(): Arrotonda un decimale per eccesso: ")

numero = 15.348

print(f"Il numero {numero} diventa {m.ceil(numero)}")

print()

print("Funzione floor(): Arrotonda un decimale per difetto: ")

numero = 1.98

print(f"Il numero {numero} diventa {m.floor(numero)}")

print()

print("Funzione trunc(): Arrotonda tutte le cifre decimali di un numero ")

numero = 77.934234323

print(f"Il numero {numero} diventa {m.trunc(numero)}")

print()

print("Funzione factoria(): Calcola il fattoriale di un numero (Es 4! = 4\*3\*2): ")

numero = 6

print(f"Il fattoriale di  {numero} è {numero}!, ossia 6\*5\*4\*3\*2 = {m.factorial(numero)}. ")

print()

print("Funzione sqrt(): Calcola la radice quadrata di un numero. Il risultato sarà un float: ")

numero = 81

print(f"La radice quadrata di {numero} è {m.sqrt(numero)}. ")

print()

print("Funzione hypot(): E' il teorema di Pitagora. Accetta due parametri: Base e Altezza di un triangolo rettangolo ")

base = 16

altezza = 3.9

print(f"Se la base è {base} e l'altezza è {altezza}, l'ipotenusa è {m.hypot(base,altezza)}")

print()

Console:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

**RANDOM MODULE**

**METODI**

**METODI RANDOM.SEED e RANDOM.RANDOM**

La funzione random genera un insieme di numeri pseudo-casuali. Per capirne il funzionamento, è necessario capire il funzionamento del metodo seed() presente nel modulo random. I numeri random saranno quindi numeri pseudo casuali, per i quali il primo numero casuale parte dal seed, e, a partire da esso, si generano n numeri pseudo-casuali compresi tra 0 e 1.

All’interno del seed si passa per argomento un intero. La stampa dei seed è sempre la stessa 🡺 La lista di numeri random che partono dal seed è sempre la stessa, in base all’indice del seed.

Per capire bene quanto detto, vediamo qualche esempio.

ESEMPIO 1: seed(7)

Codice Python:

# -\*- coding: utf-8 -\*-

"""

Created on Thu Jun  1 11:00:19 2023

@author: jacopogianfranchi

"""

import random

random.seed(7)

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print()

random.seed(7)

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print()

random.seed(7)

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print()

random.seed(7)

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

Console:

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

Esempio2: seed(55987)

Codice Python:

import random

random.seed(55987)

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print()

random.seed(55987)

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print()

random.seed(55987)

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print()

random.seed(55987)

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

Console:

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

Se il seed è sempre lo stesso e se il numero di random.random è sempre lo stesso, il numeri pseudo-casuali sono sempre gli stessi.

Come facciamo a far si che ogni serie di random.random sia diversa?

Per fortuna Python ci viene in aiuto.

Se omettiamo la funzione random.seed(), Python varia automaticamente il seed in modo da farti comparire numeri pseudo-casuali sempre diversi!!

Codice Python:

import random

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print()

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print()

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print()

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

Console:

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

Quanto visto nell’ultimo esempio equivale ad utilizzare il metodo seed() omettendo l’argomento:

Codice Python:

import random

random.seed()

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print()

random.seed()

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print()

random.seed()

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print()

random.seed()

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

print(random.random())

Console:

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

In base a cosa cambia il seed se non viene specificato manualmente? In base all’orario del proprio pc!

**METODO RANDOM.SAMPLE**

Il metodo random.sample accetta due argomenti. Entrambi gli argomenti sono obbligatori. Il primo è un iterabile, noto anche come sequenza. Come sappiamo, gli iterabili sono: stringhe, liste, tuple o set. Si ricorda che anche le chiavi e i valori dei dizionari sono iterabili.

Il secondo elemento indica quanti elementi si vogliono estrarre in modo random dall’iterabile in questione.

Il valore di ritorno della funzione random.sample è una lista, i cui numero di elementi, ossia la sua lunghezza, è pari al secondo elemento passato come argomento alla funzione.

Vedremo esempi con stringhe, liste e tuple.

Codice Python:

import random as r

anagrafica = "Alfonso Pascucci"

numeri\_lista = [3,10,44,90,15]

numeri\_tupla = 3,12,7,33,10.3,2

print(r.sample(anagrafica,2))

print(r.sample(numeri\_lista ,2))

print(r.sample(numeri\_tupla,2))

print()

print(r.sample(anagrafica,2))

print(r.sample(numeri\_lista ,2))

print(r.sample(numeri\_tupla,2))

print()

print(r.sample(anagrafica,2))

print(r.sample(numeri\_lista ,2))

print(r.sample(numeri\_tupla,2))

Console:

Immagine che contiene testo, schermata

Descrizione generata automaticamente

Quindi, nonostante il primo argomento passato alla funzione sample può essere un qualsiasi iterabile, il valore di ritorno sarà sempre una lista. Tale funzione non modifica, quindi, l’iterabile di partenza.

Se come secondo argomento di sample passassimo solo il valore 1? Ossia se volessimo estrarre solo un elemento dall’iterabile di partenza?

Il valore di ritorno sarà una lista contenente un solo elemento.

Codice Python:

import random as r

anagrafica = "Alfonso Pascucci"

numeri\_lista = [3,10,44,90,15]

numeri\_tupla = 3,12,7,33,10.3,2

print(r.sample(anagrafica,1))

print(r.sample(numeri\_lista ,1))

print(r.sample(numeri\_tupla,1))

print()

Console:

Immagine che contiene schermata, testo, Carattere

Descrizione generata automaticamente

Si noti che la lista ritornata dalla funzione sample non ha mai valori duplicati degli elementi dell’iterabile iniziale. Questo significa che il secondo argomento passato alla funzione sample deve essere un numero compreso tra 0 (In questo caso ritorna una lista composta da 0 elementi) e len(iterabile\_di\_partenza). Se la cifra passata come argomento della funzione sample è minore di 0 o maggiore di len(iterabile\_di\_partenza), tale funzione restituirà un errore.

Codice Python:

import random as r

numeri\_lista = [3,10,44,90,15]

print(r.sample(numeri\_lista,6))

Console:

Immagine che contiene testo, schermata, software, Carattere

Descrizione generata automaticamente

Finora siamo stati un po imprecisi. Abbiamo detto che la funzione sample crea una lista contenente elementi tutti diversi presenti in un iterabile di partenza. In realtà è più corretto dire che la lista ritornata dalla funzione sample contiene gli elementi presenti in posizioni diverse dell’iterabile di partenza.

Se nell’iterabile di partenza ci sono elementi uguali, è possibile quindi che nella lista di ritorno del metodo sample ci siano uno o più elementi uguali. L’importante è tenere a mente che, tali elementi, si trovano in posizioni diverse dell’iterabile di partenza. Vediamo un esempio.

Codice Python:

import random as r

nomi = ["Mario","Ubaldo","Sergio","Giuditta","Ubaldo","Mario","Raffaella","Mario"]

print(nomi)

print()

print(r.sample(nomi,4))

Console:

**Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente**

Il nome Ubaldo è stato quindi estratto due volte dal nostro iterabile di partenza. Tuttavia, nonostante il valore “Ubaldo” compaia due volte, tale valore si riferisce a due elementi diversi del nostro iterabile “nomi”.

**METODO RANDOM.CHOICE**

Questo metodo è molto simile al random.sample: Estrae un elemento dall’iterabile di partenza e, tale elemento, sarà proprio il valore di ritorno del metodo choice.

Questo metodo accetta solo un parametro, e tale parametro sarà l’iterabile dal quale si vuole estrarre l’elemento casuale. Inoltre, non modifica l’iterabile di partenza.

Codice Python:

import random as r

anagrafica = "Alfonso Pascucci"

numeri\_lista = [3,10,44,90,15]

numeri\_tupla = 3,12,7,33,10.3,2

print(r.choice(anagrafica))

print(r.choice(numeri\_lista))

print(r.choice(numeri\_tupla))

print()

print(r.choice(anagrafica))

print(r.choice(numeri\_lista))

print(r.choice(numeri\_tupla))

Console:  
Immagine che contiene schermata, testo

Descrizione generata automaticamente

Ovviamente tramite il metodo choice è possibile entrarre più valori casuali tramite un for loop:

Codice Python:

import random as r

numeri\_lista = [3,10,44,90,15]

for n in range(3):

    print(r.choice(numeri\_lista))

print()

for n in range(3):

    print(r.choice(numeri\_lista))

print()

for n in range(3):

    print(r.choice(numeri\_lista))

Console

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

Dagli output si evince un grande limite della funzione choice: essa ammette valori duplicati. Se si vuole ovviare a questo problema si può utilizzare la funzione sample vista in precedenza.

**PLATFORM MODULE**

Per capire a cosa serve il platform module, consideriamo il seguente schema:

Codice Sorgente 🡺 Python 🡺 Sistema operativo 🡺 Hardware

Il codice sorgente è il codice scritto dall’utente. Per poter funzionare, esso dovrà essere processato da Python, il quale collega il codice sorgente ed il sistema operativo. A sua volta, il sistema operativo permette al codice di raggiungere l’hardware, in modo tale che esso venga convertito in linguaggio macchina e funzioni.

Il modulo platform si utilizza per avere informazioni sul sistema operativo in uso dal pc e sull’hardware. Le funzioni del modulo platform che studieremo saranno le seguenti:

* platform.platform()
* platform.machine()
* platform.processor()
* platform.system()
* platform.python\_implementation()
* platform.python\_version\_tuple()
* platform.version()

**METODO platform.platform()**

Spiegazione.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Vediamo qualche esempio.

Codice Python:

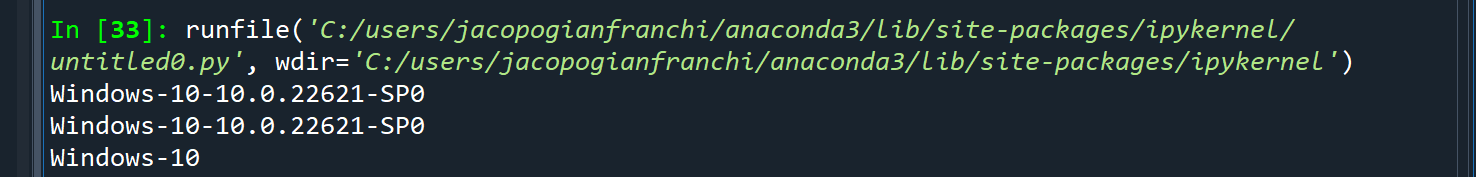
import platform as p

print(p.platform())

print(p.platform(True))

print(p.platform(True,True))

Console:



**METODO platform.machine()**

Spiegazione

**Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente**

Vediamo un esempio.

Codice Python:

import platform as p

print(p.machine())

Console:

