Brought to you by:

ASTRA

COMPUTER SCIENCE (PYTHON)

1° CLEAM/CLEF

Scritta da:
Mariachiara Alba

2023-2024 Edition

Find out more at:

astrabocconi.it

***Cos'è la programmazione.**

É istruire una macchina su come realizzare un certo compito o risolvere un certo problema. Per farlo sono necessarie precise istruzioni, ma non solo: il programmatore deve avere competenze tecniche e non tecniche.

***Ciclo di sviluppo di un programma:**

Requisiti (=obiettivi da raggiungere)

- Analisi (=analisi dei requisiti)
- Progettazione (=definizione delle linee essenziali della struttura del codice o Implementazione (=realizzazione del programma, con la programmazione) o Test : Debug (e eventualmente Progettazione)
- Esecuzione

Algoritmo = sequenza finita di azioni codificate che porta alla soluzione di un problema:

- Box ovali = blocchi di inizio e di fine o Romboide = elementi di input o output
- Rettangoli = elementi di esecuzione e rappresentano una o più azioni elementari o Rombi = elementi di decisione o scelta
- Frecce = sono il flusso di esecuzione dell'algoritmo

*** La CPU** e la memoria possono eseguire solo semplici operazioni per gestire i dati in formato macchina. I dati sono rappresentati mediante Notazione Binaria (0 e 1).

*** Programma software** = È una sequenza di comandi attraverso la quale un calcolatore esegue un'elaborazione.

*** I linguaggi di programmazione** devono essere tradotti in linguaggio macchina per essere compresi dal calcolatore. Due modalità:

Compilazione = traduzione dell'intero programma prima che venga eseguito

- Il codice viene tradotto in un linguaggio intermedio
- Esecuzione più veloce
- Interamente caricato in memoria RAM
- Regole universali per esecuzione su diverse piattaforme

Interpretazione = traduzione ed esecuzione immediata di singole istruzioni Il codice è salvato così com'è (può quindi sempre essere letto) Esecuzione più lenta

- Occupa meno spazio in memoria RAM
- Per ogni Sistema Operativo, ci può essere uno specifico «dialetto»

* Python è un linguaggio di alto livello, interpretavo, interattivo, orientato agli oggetti e open source.

* IDLE (Integrated Development and Learning Environment) è l'ambiente di programmazione incluso nell'installazione di Python ed è composto dalla shell, che permette di digitare e di eseguire immediatamente le singole line di comando, e dall'editor, che consente di creare dei programmi da salvare in memoria ed eseguire quando si desidera.

Operazioni [PEMDAS (Parentesi, Elevamento a potenza, Moltiplicazione, Divisione, Addizione,

Sottrazione)]

addizione	+	Somma due numeri
sottrazione	-	Sottrae due numeri
moltiplicazione	*	Moltiplica due numeri
divisione	/	Divide due numeri e restituisce il risultato come numero in virgola mobile
divisione intera	//	Divide due numeri e restituisce il risultato come intero (*) senza resto (per difetto)
modulo (resto)	%	Divide un numero intero per un altro e restituisce il resto della divisione
Esponente	**	Eleva un numero a una potenza

* **Funzioni built-in** (funzioni esistenti nell'installazione di Python)

print	Viene visualizzato il valore inserito tra parentesi (con " " se testo) print ('Ciao', 3, sep='*') : Ciao*3
help	Visualizza le informazioni su una funzione, su un dato o su un modulo help(format)
type	Permette di conoscere a quale tipo di dato appartiene un valore
int	Intero di dimensione arbitraria (arrotondamento per difetto) es: int(3.9): 3
float	Numero a virgola mobile
bool	Per valori veri o falsi
str	Usata per rappresentare testo
input	Funzione che chiede all'utente l'inserimento di un dato. Restituisce questo dato al programma come stringa
format	Permette di formattare i numeri, restituendo una stringa. Ha due argomenti: il numero da formattare e lo specificatore di formato: format (value, 'format_spec') ',.2f' ; ',d'

range	restituisce un oggetto che produce una sequenza di numeri interi, compresi tra il valore minimo start (incluso) e il valore massimo stop (escluso), con incrementi corrispondenti a step ATT! : per ordine decrescente: range(5, 0, -1)
sum	Calcola la somma di un. Elenco di. Elementi indicati tra parentesi quadre o tonde sperati da virgola. * Tupla: sum([1, 2, 3, 4, 5]) * Lista: sum((1, 2, 3, 4, 5)) ATT! : sum(int(3+5)) non va bene
pow	Calcola l'elevamento a potenza di un numero pow(2, 5)
abs	Calcola il valore assoluto di un numero
round	Arrotonda un numero in virgola mobile alla cifra decimale indicata come secondo parametro round(3.376, 1) : 3.4
max e min	Restituiscono il valore max o min di una serie di parametri singoli separati da virgola

*** Codici escape**

\\ print('\\')	Backslash(\) : \
\' print('\'')	Apice(') : '
\" print('\"')	Virgolette(") : "
\t	Tabulazione :
\n	A capo :

*** Tipizzazione dinamica** = Determinare il tipo di una variabile sulla base del valore che attualmente contiene

*** La struttura delle istruzioni di un programma** può essere SEQUENZIALE O DECISIONALE.

La struttura decisionale, in cui si esegue una determinata azione solo al verificarsi di condizioni specifiche, si suddivide in diversi tipi:

- Struttura decisionale semplice (o esecuzione condizionale o ad alternativa singola) Es: istruzione condizionale if
- Esecuzione alternativa (o ad alternativa doppia) Es: istruzione condizionale if-else
- Condizioni in serie

Es: istruzione condizionale if-elif (non è necessario terminare con else)

Le strutture decisionali vengono implementate nel codice per mezzo di istruzioni condizionali.

*** Operatori logici**

and	Restituisce True se tutti gli argomenti sono veri, altrimenti restituisce False
or	Restituisce True se almeno uno degli argomenti è vero, altrimenti restituisce False
not	Restituisce True se l'argomento è falso, restituisce False se l'argomento è vero

*** Costrutti iterativi**

Si dice iterazione la capacità di eseguire ripetutamente uno stesso blocco di istruzioni.
Si dice ciclo (= costrutto) una sequenza finita di istruzioni ripetuta finché il test iniziale diventa falso (while) o si raggiunge il numero di ripetizioni previsto (for).
Esistono due categorie di cicli: quelli controllati da una condizione (while) e quelli controllati da un contatore (for).

```
while condizione:
        istruzione
        istruzione
        istruzione
        ecc.
[altre linee del codice]
```

«Finché la condizione è vera, esegui le istruzioni»

ATT!: è necessario per prima cosa inizializzare la variabile e poi scrivere l'istruzione while, altrimenti uso while True che necessita delle istruzioni if e break

```
for variabile in [valore 1, valore 2, ecc.] :
        istruzione
        istruzione
        istruzione
        ecc.
[altre linee del codice]
```

«Esegui le istruzioni per ciascuno dei valori della variabile inclusi nella sequenza»

* - Si usa l'istruzione break quando si vuole terminare bruscamente il ciclo al verificarsi di determinate condizioni; - Si usa l'istruzione continue, quando, al verificarsi di determinate condizioni, è necessario passare all'iterazione successiva del ciclo saltando tutte le istruzioni seguenti nello stesso blocco - L'istruzione pass si usa quando si vuole provare il programma, o proseguire con le istruzioni seguenti, senza aver completato una certa parte; non fa nulla se non fungere da segnaposto per il codice che andrà poi completato, evitando che costrutti incompleti restituiscano un errore.

FUNZIONI

*** Funzioni personalizzate** = blocchi di codice a cui viene assegnato un nome che possono contenere a loro volta altre funzioni e che vengono eseguiti quando chiamati in maniera esplicita. Rendono un programma più chiaro ed efficiente.

*** Programma modulare** = programma in cui vi è un'unica lunga sequenza di istruzioni e in cui ogni compito viene eseguito da un'apposita funzione. Vantaggi:

- Possibilità di riutilizzazione il codice
- Maggiore semplicità e lettura del codice
- Miglioramento della fase di test e debugging

*** Sintassi generale**:

- Intestazione : prima riga composta da parola chiave def, seguita da nome istruzione, elenco parametri tra parentesi tonde e termina con due punti.
- Corpo: righe successive di istruzioni indentate, può terminare con l'istruzione return.

* ATT! : La definizione permette a Python di capire quale operazione svolga la funzione, ma non la esegue. La chiamata della funzione permette l'esecuzione della funzione.
* ATT! : I parametri sono variabili create per definire la funzione e per specificare dove e come dovranno essere utilizzati quei dati; gli argomenti sono invece i dati passati alla funzione.
* Nella funzione si possono inserire parametri obbligatori e parametri opzionali.
NB : Specificare prima tutti i parametri obbligatori, seguiti da quelli opzionali.
: Nei parametri opzionali deve essere indicato il valore predefinito da usare.

* Gli argomenti vengono passati per posizione. Gli argomenti denominati sono così definiti quando, nel momento della chiamata della funzione, si specificano gli argomenti nell'ordine che preferiamo, senza rispettare le posizioni della definizione.
ATT! : È possibile utilizzare insieme sia argomenti per posizione sia argomenti denominati, ma è necessario scrivere prima gli argomenti per posizione e poi quelli denominati, altrimenti errore.

*** Call tip** = Messaggio che compare quando si digita il nome di una funzione. Mostra la lista degli argomenti e una breve descrizione della funzione. Per attivarlo Edit --> Show Call Tip.

*** Funzione produttiva** = gruppo di istruzioni che esegue un compito specifico e quando termina restituisce un valore all'istruzione che l'ha chiamata. Termina con l'istruzione return. VS
Funzione void = gruppo di istruzioni che esegue un compito specifico ma non conserva il risultato.

*** Variabile globale** = può essere raggiunta da qualsiasi istruzione di un programma
Variabile locale = può essere raggiunta solo nel suo ambito (chiamato anche scope=parte di programma in cui la variabile risulta accessibile).

*** Stringa di documentazione** (o docstring) = commento della funzione, racchiuso tra triple virgolette e subito dopo l'intestazione della funzione. Visibile quando si legge il codice, quando si utilizza la funzione help e anche nel call tip.

*** Eccezione** = errore che si verifica durante l'esecuzione di un programma. Se prevediamo che si possa verificare un'eccezione, in Python è possibile scrivere del codice per gestirla (exception handling) Tre categorie:

- Errori di sintassi : indicano che c'è un errore nella scrittura del codice. Vi è il blocco dell'esecuzione del programma. Python mostra nella shell un messaggio di errore (messaggio di traceback) e indica il punto del codice dove si trova l'errore, ma non specifica come correggerlo. Nella Shell il messaggio di errore di solito inizia con SyntaxError. Nel file il messaggio di errore è: Invalid syntax.
- Errori di runtime : c'è un errore nel codice anche se la sintassi è corretta, quindi errori causati da operazioni che Python non può eseguire o che non comprende. Si verificano quando il programma è in esecuzione. Python mostra un messaggio di errore (solo nella shell) e indica il codice che genera l'errore e ne specifica la causa.
- Errori semantici : si verificano quando il programma viene eseguito senza generare errori, ma i risultati non sono quelli corretti perché incoerenti, non previsti o non desiderabili. Derivano da un'errata progettazione del codice (sono chiamati anche errori di logica). Utile il debugger per trovare gli errori.

* È possibile utilizzare la coppia di istruzioni try … except per prevedere e correggere eventuali eccezioni. Una o più operazioni che potrebbero generare un'eccezione vengono scritte nell'istruzione try; il codice che gestisce le eventuali eccezioni viene scritto all'interno dell'istruzione except.
ATT! : si può istruire Python affinché gestisca più eccezioni tramite l'utilizzo di alcune eccezioni specifiche:

- ZeroDivisionError = sollevato quando il divisore di una divisione è uguale a zero.
- ValueError = sollevato quando la funzione o metodo riceve un argomento del tipo di dato corretto ma con un valore che non è valido
- TypeError = sollevato quando un'operazione o una funzione sono applicate a un tipo di dato sbagliato. Questo accade per esempio quando si cerca di dividere una stringa di testo per un numero, oppure quando si cerca di indicizzare una lista con indici diversi da numeri interi
- NameError = Sollevato quando una variabile non viene trovata a livello locale o globale

* **Debugging** = gran parte del debugging avviene dopo aver scritto il codice. Oltre che ricercare, correggere gli errori è anche un processo di ottimizzazione e riscrittura del codice alla ricerca di maggiore efficienza in termini di performance, di sicurezza e di affidabilità.

STRINGHE, LISTE, TUPLE

* **Sequenze**: Le sequenze sono oggetti che contengono diversi dati, memorizzati uno dopo l'altro. Vi sono diversi tipi di sequenze: STRINGHE, LISTE, TUPLE.
* Anche se hanno caratteristiche diverse, le sequenze:
  - sono iterabili
  - possono essere mutabili (liste) o immutabili (stringhe, tuple) o la posizione di ciascun valore è identificata da un numero (indice)
  - utilizzano numerose funzioni, metodi e operazioni che consentono di accedere e lavorare sui dati.

* **Indicizzazione**: tecnica per poter accedere ai singoli elementi di una sequenza; infatti la posizione di ogni elemento all'interno di una sequenza è identificata da un numero, chiamato index.
L'indice del primo elemento a sinistra è pari a 0, mentre l'indice dell'ultimo elemento della sequenza è pari al numero di elementi della sequenza meno 1.

capitali = ['Roma', 'Tokyo', 'Berlino', 'Milano', 'Londra', 'Parigi'] print(capitali[2])
Berlino

* **Slicing**: tecnica utilizzata per selezionare più elementi di una sequenza contemporaneamente, usando gli indici degli elementi della sequenza. È possibile selezionare un intervallo all'interno della sequenza a partire dall'elemento con indice_iniziale (incluso) fino all'elemento con indice_finale (escluso).
Sequenza[indice_iniziale:indice_finale]

testo = 'PYTHON BASICS'
x = testo[0:6] y =
testo[-13:-7]
'PYTHON'

* **Metodi**: funzioni legate a un particolare oggetto. Nella sintassi oltre all'oggetto deve essere specificato il nome della sequenza.
Funzione : nome_funzione(testo)
Metodo : testo.nome_metodo(argomento)

Stringhe (immutabili)

Sequenza di caratteri che può contenere caratteri alfanumerici e simboli.
Le stringhe sono immutabili, quindi non è possibile modificare una stringa esistente: l'unica possibilità è quella di creare una nuova stringa, variante dell'originale.

Operazioni e funzioni delle stringhe:

+	Concatena più sequenze
*	Crea più copie di una sequenza e le unisce
in	Restituisce True se una stringa di testo viene trovata in una seconda stringa di testo, altrimenti restituisce False
not in	Restituisce True se una stringa di testo non viene trovata in una seconda stringa di testo, altrimenti restituisce False
is	Restituisce True se due stringhe sono identiche, altrimenti False
is not	Restituisce True se due stringhe sono diverse, altrimenti False
len(seq)	Restituisce il numero di elementi di una stringa

max(seq) e min(seq)	Restituisce il valore più grande/piccolo contenuto in una stringa (max simile a sorted). L'elemento più piccolo è lo spazio, seguito dal segno di punteggiatura poi dai numeri e infine dalle lettere (prima maiuscole e poi minuscole).
sorted(seq)	Restituisce una lista con gli elementi della stringa in ordine crescente. L'elemento più piccolo è lo spazio, seguito dal segno di punteggiatura poi dai numeri e infine dalle lettere (prima maiuscole e poi minuscole).
sum(seq)	Somma gli elementi della stringa (solo numeri)

*** Metodi delle stringhe:**

.upper()	Restituisce una copia della stringa convertita in caratteri maiuscoli.
.lower()	Contrario di .uppere()
.capitalize()	Restituisce una copia della stringa con la prima lettera in maiuscolo e tutte le altre minuscolo
.strip()	Restituisce una copia della stringa in cui tutti gli spazi all'inizio e alla fine della stringa sono stati rimossi. L'argomento car specifica il carattere da rimuovere.
.find(sub)	È "come" index. Cerca nella stringa la sottotringa specificata nell'argomento sub e
	Restituisce l'indice della prima occorrenza trovata. Se la sottostringa non viene trovata restituisce -1.
.replace(old, new)	Restituisce una copia della stringa in cui tutte le occorrenze della sottostringa old sono sostituite dalla sottostringa new.
.startswith(prefix)	Restituisce TRUE se la stringa inizia con la stringa specificata nell'argomento prefix altrimenti FALSE.
.endswith(suffix)	Restituisce TRUE se la stringa termina con la stringa specificata nell'argomento suffix altrimenti FALSE.
.count(sub)	Restituisce il numero di occorrenze della sottostringa sub nella stringa.

.split()	Divide una stringa in parole e restituisce una lista di stringhe, considerando lo spazio come separatore tra le parole. È possibile specificare due argomenti facoltativi, separator (specifica come separatore una qualsiasi stringa di testo) e maxsplit (indica il numero massimo di divisioni della stringa da effettuare). Se omessi i valori predefiniti sono rispettivamente lo spazio e -1 (ossia infinito).
.join(iterable)	Restituisce una stringa ottenuta concatenando tutti gli elementi di un oggetto iterabile contenente solo stringhe. Deve essere indicata una stringa contenente il limitatore da utilizzare, da mettere prima del punto.
.format(variable1, variable2, …)	Eg: print('{:<3} F | {:>6.2f} C'.format(x, to_celcius(x)))

LIFE-HACKS:

- ultimi 2 caratteri : [-2:]
- aggiungere un '.' (punto) finale se non è già presente : if parola[-1] != '.'
- Parola invertita (palindroma) : rev_parola = parola[::-1]
- sostituire gli spazi con il carattere "_" : testo.replace(' ','_')
- La funzione deve ricevere un numero (intero o con decimali) come parametro obbligatorio e ritornare il numero di cifre dispari contenute.

```
def a(num):
conta = 0          numero =
str(num)               for  i  in
numero:          if i != '.':
if int(i)%2==1:
conta = conta + 1
return conta
```

- Inverte il primo e ultimo carattere del testo passato come parametro (mantenere il maiuscolo/minuscolo su iniziale/finale) def a(s):

```
first = s[0]     last = s[-1]     if
first == first.upper():
ris = last.upper()     else:
ris = last.lower()      ris = ris +
s[1:-1]                if    last    ==
```

```
last.upper():          ris = ris +
first.upper()    else:
ris = ris + first.lower()    return ris
```

-estrarre la prima e la terza parola

```
spazio1 = text.find(' ')
spazio2 = text.find(' ', spazio1+1)
spazio3 = text.find(' ', spazio2+1)
#estrazione prima parola          word1 =
text[:spazio1]      #estrazione terza parola
if spazio3 > 0:
word2 = text[spazio2+1:spazio3]       else:
word2 = text[spazio1+1:spazio2]
```

Liste (mutabili)

* Una lista è una raccolta di dati (detti elementi) di qualsiasi tipo, racchiusi tra parentesi quadre e separati da virgole
* Le liste possono essere assegnate a variabili
* Una lista priva di elementi è chiamata lista vuota
* Una lista può essere nidificata all'interno di un'altra lista
* Per creare una lista è anche possibile utilizzare la funzione built-in list:

```
x = 'Bocconi' list(x)
['B', 'o', 'c', 'c', 'o', 'n', 'i']
```

*** Operazioni e funzioni delle liste:**

+	Concatena più liste in una sola
*	Crea più copie di una lista e le unisce
in	Restituisce True se un elemento è incluso in una lista, altrimenti restituisce False
not in	Restituisce True se un elemento non è incluso in una lista, altrimenti restituisce False
is	Restituisce True se due liste sono identiche, altrimenti False
is not	Restituisce True se due liste sono diverse, altrimenti False
len(seq)	Restituisce il numero di elementi di una lista
max(seq) e min(seq)	Restituisce il valore più grande/piccolo contenuto in una lista (max simile a sorted). (non possono esserci numeri e lettere insieme)

sorted(seq)	Restituisce una lista con gli elementi della lista in ordine crescente. (non possono esserci numeri e lettere insieme)
sum(seq)	Somma gli elementi della lista (solo numeri)

*** Metodi delle liste:**

.append(element)	Accoda nuovi elementi alla lista
.insert(index, element)	Inserisce l'elemento desiderato nella posizione specificata dal parametro index
.remove(element)	Cerca l'elemento specificato nella lista ed elimina la prima occorrenza
.pop([index])	Rimuove e restituisce l'elemento con indice specificato. Se si omette index rimuove l'ultimo elemento della lista.
.extend(list2)	Accoda la lista indicata (list2) alla lista
.index(element)	Cerca l'elemento indicato all'interno della lista e restituisce il suo indice. Se l'elemento compare più volte, restituisce l'indice della prima occorrenza.
.sort()	Dispone gli elementi della lista in ordine crescente. (non possono esserci numeri e lettere insieme)
.reverse()	Inverte gli elementi della lista.
.clear()	Rimuovi tutti gli elementi di una lista, restituendo una lista vuota.
.count(element)	Conta quante volte un determinato elemento compare nella lista e restituisce il numero.
.copy()	Restituisce una copia della lista.

LIFE-HACKS:

- Ordinare lista in senso decrescente my_list.sort(reverse=True)
- Creare un programma che permetta di scrivere un asterisco al posto dei numeri multipli di 2 contenuti nella seguente lista di liste: matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

```
for n in range(len(matrix)):
for i in range(len(matrix[n])):
if matrix[n][i]%2==0:
matrix[n][i]= '*'
print(matrix)
```

- creare una lista di pesi pari a 1 se l'argomento weights non viene specificato dall'utente

```
if weights == []:          weights =
[1]*len(values)
```

- Media ponderata (values, weights)

```
sumprod = 0 for i in
range(len(values)):
sumprod += values[i]*weights[i]
print(sumprod/sum(weights))
```

Tuple (immutabili)

* Gli elementi di una tupla possono essere di qualsiasi tipo, sono separati da virgole e comunemente racchiusi tra parentesi tonde (sebbene l'uso di parentesi non sia necessario) * Per creare una tuba è possibile utilizzare la funzione built-in tuple:

```
>>> x = 'Bocconi' >>> tuple(range(4)) tuple(x)    (0, 1, 2, 3)
('B', 'o', 'c', 'c', 'o', 'n', 'i')
```

* Le tuple sono immutabili, quindi non è possibile modificare una tupla esistente: l'unica possibilità è quella di creare una nuova tupla, variante dell'originale.

*** Operazioni e funzioni delle tuple:**

+	Concatena più sequenze
*	Crea più copie di una sequenza e le unisce
in	Restituisce True se un elemento è incluso in una tupla, altrimenti restituisce False
not in	Restituisce True se un elemento non è incluso in una tupla, altrimenti restituisce False
is	Restituisce True se due tuple sono identiche, altrimenti False
is not	
	Restituisce True se due tuple sono diverse, altrimenti False
len(seq)	Restituisce il numero di elementi di una tupla
max(seq) e min(seq)	Restituisce il valore più grande/piccolo contenuto in una stringa (max simile a sorted). (non possono esserci numeri e lettere insieme)

sorted(seq)	Restituisce una lista con gli elementi della lista in ordine crescente. (non possono esserci numeri e lettere insieme)
sum(seq)	Somma gli elementi della stringa (solo numeri)

*** Metodi delle tuple:**

.index(element)	Cerca l'elemento indicato all'interno della tupla e restituisce il suo indice. Se l'elemento compare più volte, restituisce l'indice della prima occorrenza.
.count(element)	Conta quante volte un determinato elemento compare nella tupla e restituisce il numero.

DIZIONARI (Non-sequenze e mutabili)

* In Python un dizionario è un oggetto che contiene una collezione di dati o elementi.
* Gli elementi sono composti da due parti: una chiave e un valore.
    - Chiavi: sono univoche, possono essere qualsiasi tipo di dato immutabile (numeri, stringhe, tuple) e sono utilizzate per trovare i valori.
    - Valori: possono essere qualsiasi tipo di dato mutabile o immutabile.
* Per creare un dizionario è necessario scrivere tra parentesi graffe { } gli elementi separati da virgole (,). Ogni elemento è costituito da una chiave seguita da due punti (:) e da un valore storage = {'prod1': 425, 'prod2':45, 'prod3': 564, 'prod4': 213}

* In alternativa è possibile utilizzare la funzione dict, che crea un nuovo dizionario privo di elementi.
* I dizionari sono oggetti mutabili.
* Gli elementi di un dizionario non sono memorizzati in un ordine particolare, non è quindi possibile utilizzare l'indicizzazione come nelle sequenze.
* Per estrarre un valore da un dizionario si utilizza la chiave associata al valore:
nome_dizionario[chiave]
* Per aggiungere coppie chiave-valore: nome_dizionario[chiave] = valore

*** Operazioni e funzioni dei dizionari:**

in	
	Restituisce True se un elemento (chiave) è incluso in un dizionario, altrimenti restituisce False
not in	Restituisce True se un elemento (chiave) non è incluso in un dizionario, altrimenti restituisce False

is	Restituisce True se un dizionario è identico a un altro, altrimenti False
is not	Restituisce True se un dizionario è divrso da un altro, altrimenti False

ATT! Gli operatori sono utilizzati per sapere non solo se i due dizionari sono identici, ma se occupano la stessa zona memoria. Infatti, se viene modificata una copia chiave-valore in un dizionario, la stessa modifica viene applicata anche all'altro dizionario.

```
storage5 = {'prod8': 51, 'prod4': 305}
storage6 = storage5
storage5 is storage6
True
{'prod8': 100, 'prod4': 305}
```

```
storage5[ 'prod8'] = 100
    storage5
 {'prod8': 100, 'prod4': 305}
    storage6
```

len(seq)	Restituisce la lunghezza di un dizionario, cioè il numero di coppie chiave-valore del dizionario.
max(seq) e min(seq)	Restituisce la chiave più grande/piccola del dizionario. Quando le chiavi sono tutte delle stringhe : L'elemento più piccolo è lo spazio, seguito dal segno di punteggiatura poi dai numeri e infine dalle lettere (prima maiuscole e poi minuscole). Se le chiavi non sono tutte stringhe, le chiavi devono appartenere alla stessa classe, altrimenti errore
sorted(seq)	Restituisce una lista con le chiavi del dizionario ordinate in maniera crescente. Quando le chiavi sono tutte delle stringhe : L'elemento più piccolo è lo spazio, seguito dal segno di punteggiatura poi dai numeri e infine dalle lettere (prima maiuscole e poi minuscole). Se le chiavi non sono tutte stringhe le chiavi devono appartenere alla stessa classe, altrimenti errore

*** Metodi dei dizionari:**

.clear()	Rimuove tutti gli elementi di un dizionario, ma non elimina il dizionario.
.get(key,[default])	
	Restituisce il valore associato alla chiave specificata dall'argomento key. Se la chiave specificata non è presente nel dizionario restituisce il valore None oppure restituisce il valore specificato nell'argomento default.
.pop(key[default])	Rimuove la chiave specificata nell'argomento key (e il valore associato) e restituisce il valore associato. Se la chiave specificata non è presente e non è stato impostato un valore predefinito nell'argomento default, restituisce l'errore KeyError.
.popitem()	Rimuove l'ultima coppia chiave-valore aggiunto al dizionario e restituisce una tupla con due elementi (la chiave e il valore rimossi)
.update(dict2)	Aggiunge le chiavi e il valore di un secondo dizionario (dict2) a un dizionario esistente
.items()	Crea un oggetto di tipo dict_items che contiene una lista di tuple, ciascuna con due elementi (le coppie chiave-valore)
.keys()	Crea un oggetto di tipo dict_keys che contiene una lista, in cui ogni elemento è una delle chiavi del dizionario.
.values()	Crea un oggetto di tipo dict_valuees che contiene una lista, in cui ogni elemento è uno dei valori del dizionario.

LIFE-HACKS:

- leggere il contenuto del file Esercizio10.3_testo.txt e salvarlo nella variabile text

```
file = open("Esercizio10.3_testo.txt", 'r') text =
file.read()
```

- partendo da una tupla composta dalle vocali a-e-i-o-u, creare il dizionario nVocali in cui salvare le vocali come chiavi e quante volte sono presenti in text come valori

```
vocals = ('a','e','i','o','u')
nVocali = {} text.lower() for i in
vocals: nVocali[i] =
text.count(i)
print(nVocali)
```

ACCESSO AI FILE

* Per accedere a un file in Python si utilizza la funzione open.
    - Apre un file e restituisce un oggetto di tipo file al quale viene associato il contenuto dello specifico file che è stato aperto
    - Se il file non può essere aperto, viene sollevato un errore
    - L'apertura di un file comporta la creazione di una variabile a cui associare l'oggetto

: file_dati = open(file, mode)

Modalità

'r'	Apre il file in modalità sola lettura. Il puntatore è posizionato all'inizio del file
'w'	Apri il file in modalità scrittura. Se il file esiste già i dati contenuti vengono eliminati. Il puntatore è posizionato all'inizio del fine
'a'	Apri il file in modalità aggiunta o accodamento. Se il file esiste già i dati contenuti vengono aggiunti alla fine del documento

*** Leggere dati da un file:**

Metodi

.read(size)	Leggere il contenuto del file dalla posizione corrente del contatore fino alla fine. L'argomento size è opzionale e specifica il numero di byte da leggere.
.readline(size) .readlines(size)	Legge e restituisce in una stringa il contenuto di una riga del file. L'argomento size è opzionale e specifica il numero di byte da leggere. Legge tutte le righe e restituisce una lista contenente tutte le righe lette come elementi.

*** Scrivere dati in un file:**

Metodi

.writable()	Restituisce TRUE se è possibile scrivere dati nel file, altrimenti restituisce FALSE.
.write(string)	Scrive nel file il contenuto dell'argomento string e restituisce il numero di caratteri inseriti. Per concludere una riga nel file di testo, occorre scrivere il carattere "new line". : f.write("Salvo Brandi\n")

ALTRI MODULI DELLE LIBRERIE PYHTON

* Qualunque linguaggio di programmazione può essere arricchito da estensioni per svolgere funzionalità aggiuntive o specifiche.

* In Python, le funzionalità aggiuntive vengono fornite tramite i MODULI: si tratta di file script che fungono da contenitori, raggruppando le funzionalità per tema, svolgendo azioni o calcoli specifici.

* Alcuni sono presenti nella installazione di base di Python, e ne conosciamo già alcuni esempi:
math, random.

* I moduli possono essere organizzati in LIBRERIE o «package».

* I moduli installati di default con Python costituiscono la libreria standard di Python.

* Cosa si trova in un modulo?

- Funzioni
- Tipi di dati speciali o Metodi: funzioni costruite per lavorare specificatamente con le loro caratteristiche (dette attributi) di un certo tipo di dato speciale. o Doctring o Commenti
- Dati di esempio

* Per poter utilizzare un modulo è necessario importarlo con l'istruzione import.
NB! L'importazione vale solo per la sessione corrente, siccome il modulo viene caricato nella RAM.

* L'istruzione from permette di importare una funzione di un modulo senza necessariamente importare l'intero modulo.

* Per sapere quali moduli sono disponibili in una sessione di lavoro nella shell di Python, utilizzo il seguente comando : dir()

* Se volessimo avere un elenco delle funzioni contenute in un modulo, si può utilizzare la funzione dir(modulo).

from math import sqrt, ceil *

Funzioni modulo math:

math.ceil(x)	Restituisce l'intero arrotondato per eccesso di X

math.floor(x)	Restituisce l'intero arrotondato per difetto di X
math.sqrt(x)	Restituisce la radice quadrata di X
math.pi(x)	Restituisce il valore del pi greco
math.exp(x)	Restituisce Il numero di Nepero elevato alla potenza X.
math.pow(x,y)	Funziona come la funzione pow, che eleva X alla Y
math.factorial(x)	Restituisce il fattoriale del numero intero positivo X.
math.gdc(x,y)	
	Restituisce il massimo Comun divisore tra i due numeri interi X e Y.
math.hypot(x,y)	Applica la formula di Pitagora per trovare l'ipotenusa tramite i due cateti X e Y.
math.log(x,[baseA])	Restituisce il logaritmo in base A di X. Se A messa calcola il logaritmo naturale.

*** Modulo os (per interagire con il Sistema Operativo):**

- Permette a Python di interagire con il Sistema Operativo, particolarmente utile per ispezionare e manipolare file e cartelle. o Funzioni:

os.listdir([path])	Restituisce una lista di nomi dei file e delle cartelle presenti nella cartella, indicata tramite il percorso path.
os.getcwd()	Restituisce una stringa che rappresenta il percorso dell'attuale working directory.
os.rename(oldName, newName)	Rinomina il file o la directory.
os.remove(file)	Cancella il file.
os.rmdir(path)	Cancella la directory, ma solo se è vuota.
os.mkdir(path)	Crea una nuova directory; se già esistente da errore.
os.chdir(path)	Fa diventare il percorso path la nuova working directory.
os.path.join(path, filename)	Restituisce il percorso completo di un file in una sola cartella specificata.
os.path.isfile(path)	Restituisce TRUE se il percorso fornito identifica un file, altrimenti FALSE.

*** Modulo random:**

random.random()	Genera un numero casuale decimale tra 0 e 1;
random.randint(a, b)	Restituisce un numero intero casuale dell'intervallo (estremi inclusi)
random.randrange(a,b+1)	Fenera numeri casuali appartenenti all'intervallo. ES : randrange(8) = randrange(0,7)
os.remove(file)	Cancella il file.
os.rmdir(path)	Cancella la directory, ma solo se è vuota.
os.mkdir(path)	Crea una nuova directory; se già esistente da errore.

*** Modulo webbrowser:**

Permette a Python di aprire un URL nel browser. : webbrowser.open(URL)

INSTALLARE MODULI PERSONALIZZATI : * Per usare un modulo personalizzato è necessario scaricarlo dal sito e installarlo seguendo le istruzioni, oppure, più semplicemente eseguendo pip install modulename dalla riga di comando del sistema operativo:

- Scriviamo cmd.exe nel box di ricerca del menu Windows (in MacOS, apriamo l'applicazione Terminal)
- Nella riga di comando, scriviamo pip install modulename.

CLASSI, ATTRIBUTI E METODI

CLASSI

* Python supporta diversi tipi di dati.
* A ogni variabile può essere assegnato un valore di diverso tipo, ma una variabile può anche essere usata per eseguire un'azione.
* Le azioni che ogni variabile può fare dipendono dal tipo di dato a cui appartiene.
* Nei linguaggi orientati agli oggetti (come Python) gli elementi che possono rappresentare un valore o eseguire azioni sono definiti come Oggetti -> le variabili sono tutte degli oggetti.

* Ogni tipo di variabile ha caratteristiche diverse sia nei valori che può assumere sia nelle azioni che si possono eseguire con essa
* Tutte le variabili di un certo tipo possono eseguire le stesse azioni.
* Nei linguaggi orientati agli oggetti (come Python) l'insieme delle caratteristiche che può assumere un determinato tipo di variabile è definito una Classe .

*** Classe:** descrizione astratta di un oggetto, la famiglia o l'insieme di oggetti di un certo tipo, che possono essere predefiniti in Python o costruiti nelle librerie di terze parti o da no i.
*** Istanza**: indica un caso specifico di una classe, un certo insieme di valorizzazioni degli attributi della classe. Infatti, quando si inizializza una variabile viene creato uno specifico Oggetto di una specifica Classe.

ATTRIBUTI E METODI

* Una classe è definita da un nome e da un insieme di attributi e metodi
* Gli attributi sono le caratteristiche della classe:
  -il valore di una stringa è un suo attributo : string ha l'attributo «valore»
* I metodi sono le «azioni» che gli oggetti di una classe possono compiere:
  -upper : è un metodo della classe string o append : è un metodo della classe list

RIASSUMENDO

* Oggetto: è il termine astratto utilizzato per «elementi» caratterizzati da attributi (come è fatto) e metodi (come possiamo modificarlo e usarlo)
* Classe: è la descrizione astratta di un oggetto. Rappresenta l'insieme di oggetti (la «famiglia») di un certo tipo. Una classe è costituita da attributi e metodi
* Istanza: indica un particolare oggetto «reale» di una classe

Attributi della classe

- Consentono di personalizzare i singoli oggetti della classe (i.e. le istanze)
- Possono essere tipi built-in di Python (es.: int, float, list) o altri oggetti
- Si possono definire direttamente nella classe (con una semplice assegnazione, come mostrato qui sotto), ma sono solitamente definiti con uno speciale metodo definito metodo costruttore.

Metodo costruttore __init__

- È il metodo che Python cerca nella classe quando deve inizializzare lo stato di un oggetto.
- Può includere la definizione degli attributi e di qualsiasi altra caratteristica e opzione necessaria agli oggetti della classe, come: chiamata di funzioni, apertura di file, accesso ai database ecc.
- La sintassi segue le stesse regole utilizzate durante la definizione di una funzione, con i parametri (obbligatori o opzionali) che vengono utilizzati per inizializzare gli attributi, specificati tra parentesi
- Deve essere presente almeno un parametro, convenzionalmente denominato self, che rappresenta il riferimento al nome dell'istanza che verrà creata.

Interagire con gli attributi degli oggetti

- Per accedere a uno specifico oggetto bisogna utilizzare il nome della relativa istanza • Per definire il valore di un attributo si può utilizzare:

  -NomeIstanza = NomeClasse(val_1, val_2, ...)

(solo quando l'attributo è stato definito come parametro del metodo costruttore
__init__) o NomeIstanza.NomeAttributo = valore
(per attributi non definiti come parametri. Ma può essere utilizzato anche per gli attributi definiti come parametri nel metodo __init__)

: mario.nome -------> 'Mario'
: print(mario.nome) -------> Mario
: mario.IMC() -------> 24. 99912468050839

Metodo speciale __str__

* Cosa accade se utilizziamo il nome di un'istanza come argomento di un comando print?
* Ciò che viene visualizzato è il risultato del metodo speciale __str__
* Non ha argomenti (oltre a self) e deve terminare con l'istruzione return

Gerarchie ed Ereditarietà

* Parent class (sovraclasse)
* Child class (sottoclasse)
    - Eredita tutti i dati e i metodi della parent class
    - Aggiunge più informazioni e metodi
    - Sovrascrive metodi

* Ereditarietà: meccanismo di costruzione delle classi che permette di costruirne di nuove a partire da altre esistenti, in una vera e propria gerarchia in cui a valle (le classi figlie) ereditano attributi e metodi di quelle a monte (le classi base), con ovvie possibilità di riuso del codice già scritto.

* Polimorfismo: meccanismo secondo il quale si può dare il medesimo nome a metodi che compiono lo stesso tipo di azione astratta su oggetti differenti e sta poi al programma capire, in base al tipo di dati con cui sta operando, come compiere l'azione.