

PYTHON

è un linguaggio di programmazione **CASE SENSITIVE**

Commenti	# (hash o cancelletto) # programma di calcolo
Variabili Il linguaggio Python utilizza la tipizzazione dinamica dei dati, cioè il tipo di dato di una variabile è determinato dal valore assegnato alla variabile stessa in fase di inizializzazione.	Una variabile è definita assegnandole un valore (inizializzandola). Se non viene inizializzata produce il seguente messaggio: NameError: name 'variabile' is not defined Es.: >>> a = 10 >>> a 10 >>> b= 3 >>> a + b >>> 13 >>> a*b >>>30 >>>b/c NameError: name 'c' is not defined >>> n NameError: name 'n' is not defined
Operatori aritmetici	+ addizione - sottrazione * moltiplicazione / divisione reale // divisione intera % resto della divisione intera ** potenza di un numero
Espressioni	Le espressioni di calcolo sono costruite usando le parentesi tonde per indicare i calcoli parziali. Es.: >>> a= 15 >>> b= 2 >>> a+b 17 >>> a - b 13 >>> a* b 30 >>> a / b 7.5 >>> a // b 7 >>> a% b 1 >>> a ** b 225

	<pre>>>> (a+b) ** 2 289 >>> a * b + 1 31 >>> a * (b + 1) 45</pre>
Operatore di assegnazione	<p>= assegna il valore ad una variabile.</p> <p>Es.:</p> <pre>>>> x = 3 >>> y = 2 * x - 1 >>> y 5</pre>
Assegnazione multipla	<p>Esistono due modalità:</p> <ul style="list-style-type: none"> • Assegnazione dello stesso valore a diverse variabili • Assegnazione di valori diversi a variabili diverse (separati da virgola). <p>Es.:</p> <pre>>>> a = b = c = 9 >>> a 9 >>> b 9 >>> c 9 >>> a, b = 10, 20 >>> a 10 >>> b 20 >>> x, y = 3, 8 >>> x 3 >>> y 8 >>> a, b = x, x + y >>> a 3 >>> b 11</pre>
Osservazione	<p>Nella modalità interattiva si può usare anche una particolare variabile rappresentata dal simbolo _ (underscore) che indica il valore dell'ultima variabile (o espressione) visualizzata</p> <p>Es.:</p> <pre>>>> prezzo = 2000 >>> prezzo / 100 * 10 200.0 >>> prezzo - _ 1800.0</pre>

<p>Tipi di dati numerici</p>	<p>Python utilizza il tipo int per i numeri interi e il tipo float per i numeri con la parte frazionaria (numeri decimali in virgola mobile). Nelle espressioni di calcolo con operandi di diverso tipo, i numeri interi sono automaticamente convertiti in float.</p> <p>Es.:</p> <pre>>>> 3 + 18.2 * 2 39.4</pre>
<p>Numeri complessi</p>	<p>I numeri complessi sono rappresentati indicando il suffisso j per la parte immaginaria. Per ottenere la parte reale e la parte immaginaria di un numero complesso <i>z</i> si usano le notazioni <i>z.real</i> e <i>z.imag</i>.</p> <p>Es.:</p> <pre>>>> z = 4 + 5j >>> z (4 + 5j) >>> z.real 4.0 >>> z.imag 5.0</pre> <p>La notazione con il punto indica che <i>real</i> e <i>imag</i> sono attributi dell'oggetto <i>z</i> di tipo complesso</p>
<p>Operatori di confronto</p>	<p>< minore di > maggiore di == uguale a #rappresenta un confronto <= minore uguale a >= maggiore uguale a != diverso da</p> <p>Es.:</p> <pre>>>> a= -3 >>> b = 5 >>> a < b True >>> a > b False >>> a == b False >>> a <= b True >>> a >= b False >>> a != b True</pre>
<p>Stringhe</p>	<p>Le stringhe sono sequenze di caratteri delimitate da apici singoli oppure da virgolette. Le stringhe sono immutabili, cioè non possono essere cambiate nel loro</p>

	<p>valore.</p> <p>Es.:</p> <pre>>>> s1 = "Rossi" >>> s2 = "Bianchi"</pre>
Carattere di escape \ (backslash)	<p>Per eliminare il significato di metacarattere dell'apice si possono usare le virgolette come delimitatore, oppure si aggiunge il carattere escape \ (backslash) prima dell'apice. Lo stesso vale per le virgolette.</p> <p>Es.:</p> <pre>>>> s3 = 'L\'elenco delle persone' >>> s4 = "L'elenco delle persone" >>> print(s3) L'elenco delle persone >>> print (s4) L'elenco delle persone</pre>
Operatori per le stringhe	<p>+ concatenazione di stringhe * ripetizione di una stringa</p> <p>Es.</p> <pre>>>> s5= 'auto' >>> s6= 'mobile' >>> s5 +s6 'automobile' >>> s5 *4 'autoautoautoauto'</pre>
Uso di caratteri della codifica Unicode	<p>Il linguaggio Python consente l'uso di caratteri della codifica Unicode tramite la combinazione di caratteri \u (detta Unicode-Escape), che precede il codice di un carattere.</p> <p>Es.:</p> <p>00A9 indica il simbolo copyright</p> <pre>>>> print('Questo documento è protetto da\u00A9 Copyright') Questo documento è protetto da © Copyright</pre> <p>2122 indica il simbolo trademark</p> <pre>>>> print (' Marchio registrato \u2122') Marchio registrato ™</pre>
L'input dei dati	<p>Per acquisire dati dallo standard input (tastiera) si usa la funzione input() che restituisce una stringa con i caratteri immessi dall'utente.</p> <p>Per facilitare l'interazione con l'utente, è sempre utile inviare un messaggio, racchiuso tra virgolette o tra apici singoli, per ricordare il tipo di dato da inserire.</p> <p>Es.:</p> <pre>>>> nome= input () Angela >>> print(nome) Angela</pre> <p>Per ottenere il valore numerico di una stringa contenente cifre occorre applicare la funzione int() oppure float(), al dato restituito dalla funzione</p>

	<code>input()</code> .
Casting : Operazione di conversione di un dato da un tipo ad un altro tipo.	<p>Es.:</p> <pre>>>> lato = int(input("inserisci il lato del quadrato: ")) inserisci il lato del quadrato: 4 >>> area= lato ** 2 >>> print(area) 16</pre> <p>Es.:</p> <pre>>>> r= float(input("Inserisci il raggio: ")) Inserisci il raggio: 10.2 >>> a= r ** 2 * 3.14 >>> print(a) 326.68559999999997</pre>
Osservazione	Il valore di pi greco si ottiene in Python con la costante math.pi , definita nella libreria standard <i>math</i> .
L'output dei dati	<p>La funzione print() visualizza una stringa oppure un valore numerico sullo standard output (video). Dopo la scrittura il cursore viene riportato automaticamente a capo. Per scrivere messaggi e variabili, oppure più variabili sulla stessa riga, si usa la virgola.</p> <p>Es.:</p> <pre>>>> print("L'area del quadrato di lato", lato, "è", area) L'area del quadrato di lato 4 è 16 >>></pre> <p>La virgola inserisce automaticamente uno spazio di separazione tra messaggio e valore della variabile.</p>

Coding	
Calcolare l'area del triangolo date le misure della base e dell'altezza.	
I: b, h O: a	<p>Legenda:</p> <p>b= base del triangolo h= altezza del triangolo a= area del triangolo</p>
Inizio (calcola) Chiedo, scrivo b Chiedo, scrivo h $a = b \cdot h / 2$ visualizza a fine	<p>CODIFICA PYTHON:</p> <pre>#triangolo.py: area del triangolo # input dei dati b = int(input("misura della base: ")) h = int(input("misura dell'altezza: ")) # calcolo dell'area a = b * h / 2 # output del risultato print("la misura dell'area è:", a)</pre>

```
===== RESTART: C:\Users\Studente\Desktop\ex con Python\ex_area_triangolo.py =====
misura della base: 4
misura dell'altezza: 5
la misura dell'area è: 10.0
>>>
```

Gli errori di programmazione Possono essere dei seguenti tipi:

Lessicali	Consistono nell'uso di termini non appartenenti al linguaggio.
Sintattici	Consistono nella costruzione di frasi non corrette dal punto di vista delle regole grammaticali del linguaggio.
Logici (o di semantica)	Riguardano la correttezza dell'algoritmo: il programma non produce i risultati attesi. Se un programma traduce un algoritmo sbagliato ma è scritto nel rispetto delle regole del linguaggio, l'interprete Python non può accorgersi dell'errore.
Runtime	Si possono verificare durante l'esecuzione del programma, sulla base di particolari valori assunti dai dati durante l'elaborazione: per esempio, nella divisione, quando il divisore assume il valore 0 durante l'esecuzione.

Coding

Modificare il testo di un programma Python introducendo artificialmente degli errori.

Consideriamo il seguente programma Python che calcola il perimetro di un quadrato dopo aver acquisito da tastiera la misura del lato.

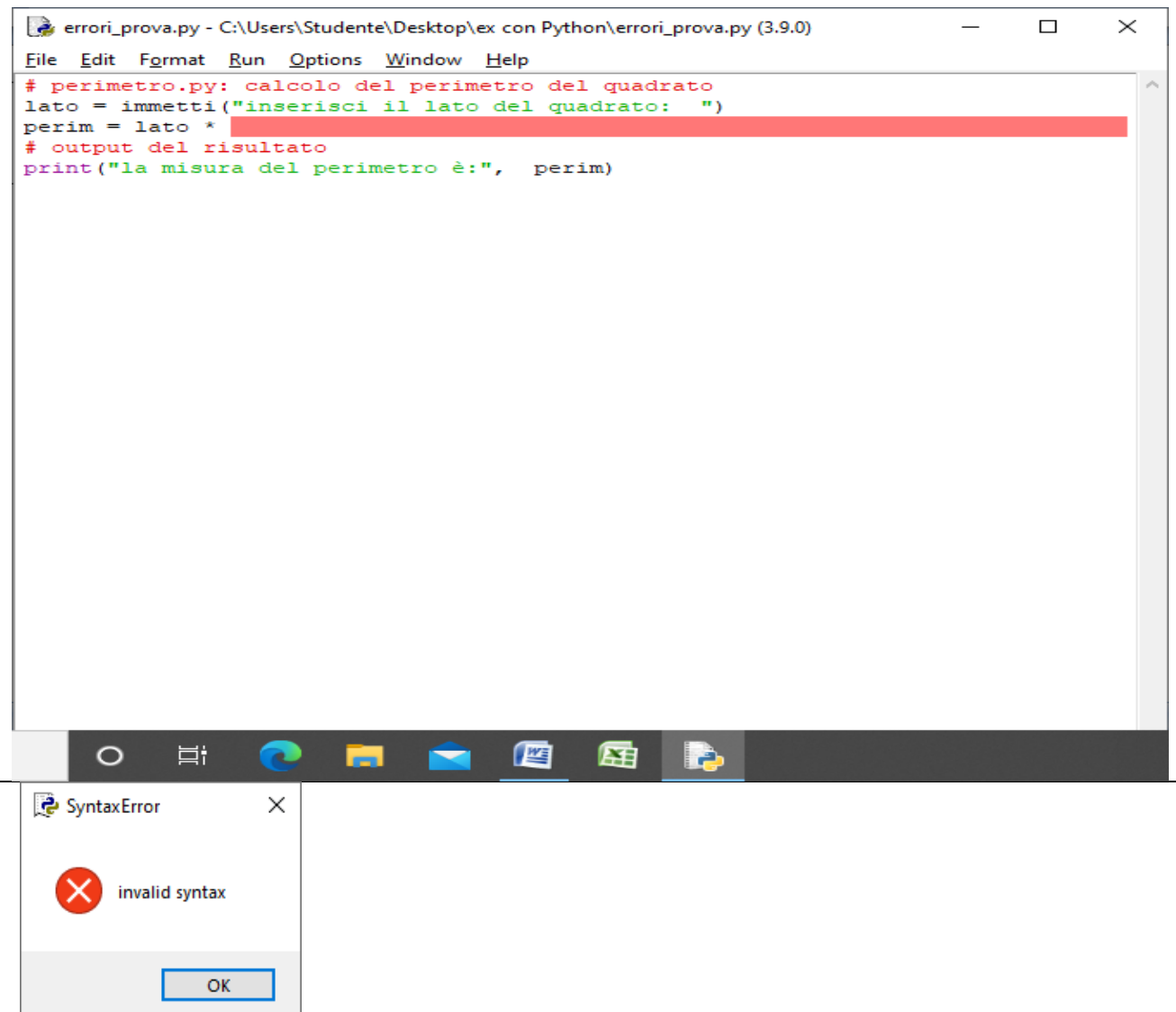
I: lato O: perim	Legenda: lato= lato del quadrato perim= perimetro del quadrato
Inizio (calcola) Chiedo, scrivo lato perim = lato*4 visualizza perim fine	CODIFICA PYTHON: # perimetro.py: calcolo del perimetro del quadrato lato = int(input("inserisci il lato del quadrato: ")) perim = lato * 4 # output del risultato print("la misura del perimetro è:", perim)

Si inseriscono in modo artificioso tre errori nel codice:

1. L'eliminazione della funzione *int* prima di *input*;
2. La sostituzione di *input* con la parola *immetti*;
3. L'eliminazione del numero 4 nella terza riga.

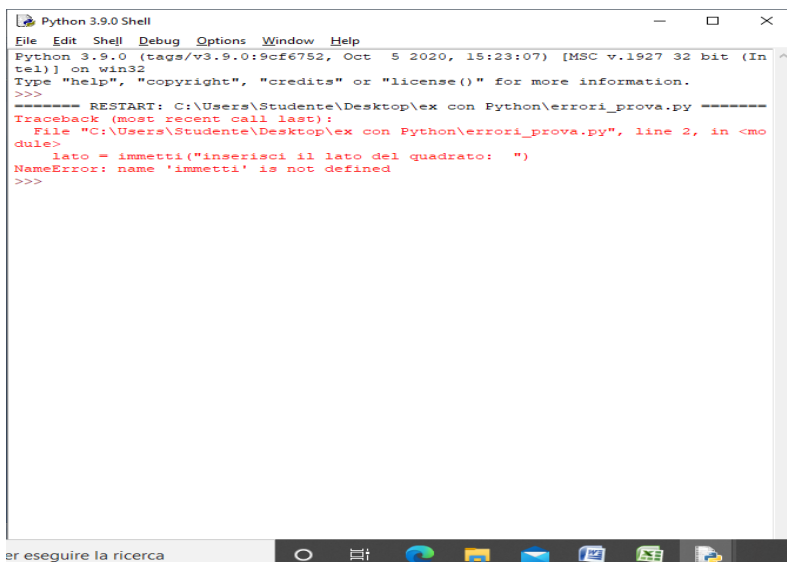
```
# perimetro.py: calcolo del perimetro del quadrato
lato = immetti("inserisci il lato del quadrato: ")
perim = lato *
# output del risultato
print("la misura del perimetro è:", perim)
```

Quando si avvia il controllo del codice (scelta Check Module del menu Run), l'interprete segnala un **errore sintattico** nella riga 3, perché manca il secondo fattore della moltiplicazione.



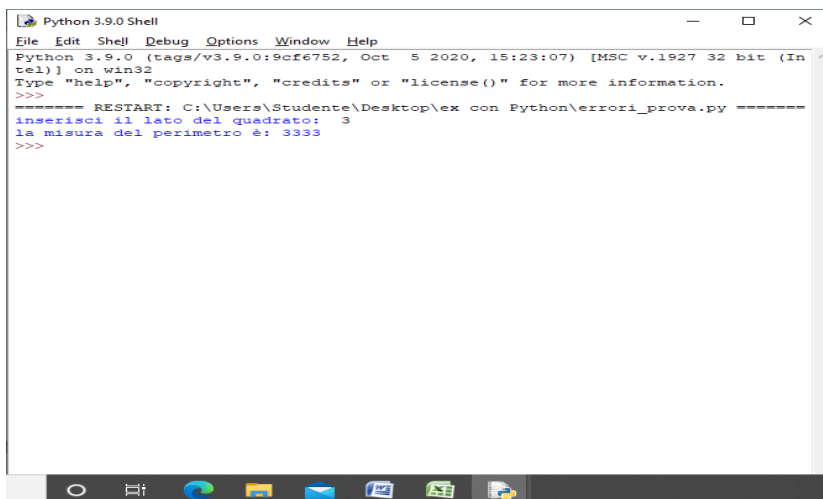
Dopo avere corretto l'errore, inserendo il numero 4 come secondo fattore, si provi a eseguire il programma (scelta Run Module nel menu Run).

L'esecuzione, nella finestra della Shell si interrompe alla linea 2, perché l'interprete ha incontrato la parola *immetti* che non appartiene al linguaggio. Questo è un **errore lessicale** in quanto è stata utilizzata una parola che non esiste nel dizionario del linguaggio Python.



```
Python 3.9.0 Shell
File Edit Shell Debug Options Window Help
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct 5 2020, 15:23:07) [MSC v.1927 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\Studiante\Desktop\ex con Python\errori_prova.py =====
Traceback (most recent call last):
  File "C:\Users\Studiante\Desktop\ex con Python\errori_prova.py", line 2, in <module>
    lato = immetti("inserisci il lato del quadrato: ")
NameError: name 'immetti' is not defined
>>>
```

Si corregga la parola immetti sostituendola con input. Rieseguendo il programma si ottiene:



```
Python 3.9.0 Shell
File Edit Shell Debug Options Window Help
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct 5 2020, 15:23:07) [MSC v.1927 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\Studiante\Desktop\ex con Python\errori_prova.py =====
inserisci il lato del quadrato: 3
la misura del perimetro è: 3333
>>>
```

Il risultato è diverso da quello che ci aspettiamo (12).

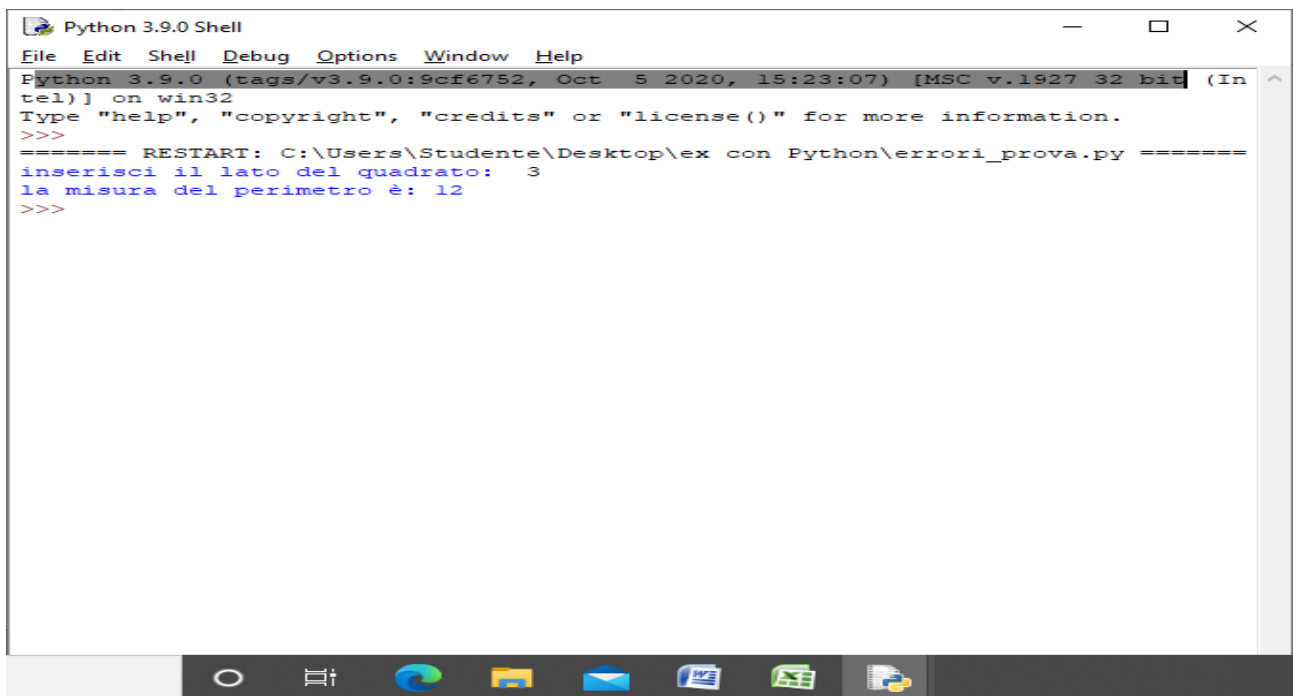
Questo è dovuto ad un **errore logico** del codice: non è stata convertita in numero intero la stringa acquisita da tastiera con la funzione input ed il programma ha applicato l'operatore di ripetizione * sulla stringa, scrivendo 4 volte la cifra 3.

Gli errori logici riguardano la correttezza dell'algoritmo e non sono rilevabili dall'interprete che controlla solo la correttezza formale del codice: è compito del programmatore controllare che l'output dell'elaborazione corrisponda a risultati corretti. Questa fase del lavoro si chiama **validazione** del codice.

La riga per l'input dei dati deve essere quindi corretta:

lato = int(input("inserisci il lato del quadrato: "))

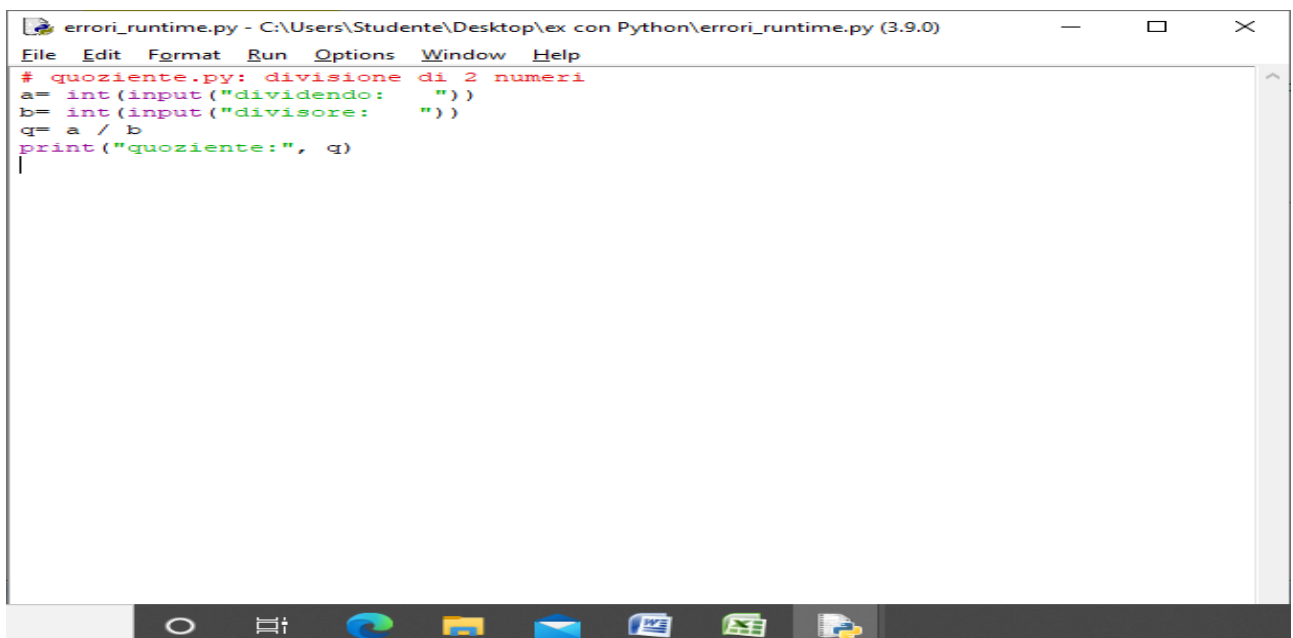
dopo questa modifica, il programma viene eseguito correttamente:



```
Python 3.9.0 Shell
File Edit Shell Debug Options Window Help
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct 5 2020, 15:23:07) [MSC v.1927 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\Studiante\Desktop\ex con Python\errori_prova.py =====
inserisci il lato del quadrato: 3
la misura del perimetro è: 12
>>>
```

Gli errori di runtime

Oltre agli errori di tipo lessicale, sintattico e logico, che sono riscontrati nella fase di collaudo del codice, nell'attività di programmazione si possono incontrare errori che vengono segnalati durante l'esecuzione del programma (**errori di runtime**) e che bloccano l'avanzamento del processo di elaborazione.



```
errori_runtime.py - C:\Users\Studiante\Desktop\ex con Python\errori_runtime.py (3.9.0)
File Edit Format Run Options Window Help
# quoziente.py: divisione di 2 numeri
a= int(input("dividendo: "))
b= int(input("divisore: "))
q= a / b
print("quoziente:", q)
|
```

Proviamo ad eseguire il programma una prima volta inserendo come divisore 30 e come dividendo 5; il risultato sarà quello che ci aspettiamo e cioè 6.

Proviamo ad eseguire il programma una seconda volta inserendo come divisore 30 e come dividendo 0; il programma si interrompe e genera questo errore:

```

Python 3.9.0 Shell
File Edit Shell Debug Options Window Help
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct 5 2020, 15:23:07) [MSC v.1927 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\Studiante\Desktop\ex con Python\errori_runtime.py =====
dividendo: 30
divisore: 0
Traceback (most recent call last):
  File "C:\Users\Studiante\Desktop\ex con Python\errori_runtime.py", line 4, in <module>
    q= a / b
ZeroDivisionError: division by zero
>>>

```

Il programma si blocca perché ha incontrato una situazione di errore durante l'esecuzione (ovvero una divisione per zero).

<p>Caratteristiche generali del linguaggio Python:</p> <ul style="list-style-type: none"> • è multiplatforma; • utilizza diversi paradigmi di programmazione; • utilizza la tipizzazione dinamica; • sono disponibili in modo predefinito numerose strutture dati di alto livello; • i programmi possono essere suddivisi in moduli; 	<p>Il nome del linguaggio deriva dallo show televisivo di sketch comici della BBC Monty Python's Flying Circus ("Il circo volante dei Monty Python").</p> <p>Python è stato ideato da Guido van Rossum un informatico olandese. Python possiede una licenza open source, che lo rende liberamente utilizzabile e distribuibile anche per uso commerciale. La licenza e lo sviluppo di Python sono amministrati dalla <i>Python Software Foundation</i>.</p> <p>Python è un linguaggio interpretato, cioè il programma sorgente non viene compilato ma eseguito dall'interprete. Da questo punto di vista può essere definito anche un linguaggio di scripting</p> <p>Il termine script in informatica indica, in generale, un testo contenente una sequenza di comandi, istruzioni e strutture di selezione if...else o di ripetizione, che è eseguita da un programma interprete.</p>
--	--

<ul style="list-style-type: none"> fornisce un insieme di librerie standard . 																			
Tipi di dato	<p>Python usa la tipizzazione dinamica dei dati, cioè il tipo di dato di una variabile è determinato dal valore assegnato alla variabile stessa in fase di inizializzazione.</p> <p>I programmi Python operano su oggetti: una variabile è il nome di un oggetto che viene associato all'oggetto con l'istruzione di assegnamento.</p> <p>Il dato è un oggetto definito dalla classe (class)</p> <p>Di seguito vengono elencati i principali tipi di dato di Python con la relativa class, cioè la categoria di valori che il dato può assumere:</p> <table> <tr> <th>Tipo di dato</th><th>Class</th></tr> <tr> <td>Intero</td><td>int</td></tr> <tr> <td>Reale in virgola mobile</td><td>float</td></tr> <tr> <td>Complesso</td><td>complex (parte reale e parte immaginaria)</td></tr> <tr> <td>Booleano</td><td>boolean (assume solo i valori True o False)</td></tr> <tr> <td>Stringa</td><td>str</td></tr> </table> <p>Oss.:</p> <p>I numeri reali in virgola mobile possono essere scritti con il punto decimale (virgola fissa) oppure in notazione esponenziale.</p> <p>Es.</p> <table> <tr> <td>.678</td><td>equivale a 0.678</td></tr> <tr> <td>2.45e34</td><td>equivale a 2.45×10^{34}</td></tr> <tr> <td>.56e-12</td><td>equivale a 0.56×10^{-12}</td></tr> </table> <p>La funzione type()</p> <p>La funzione type() restituisce la classe (class) di un oggetto.</p>	Tipo di dato	Class	Intero	int	Reale in virgola mobile	float	Complesso	complex (parte reale e parte immaginaria)	Booleano	boolean (assume solo i valori True o False)	Stringa	str	.678	equivale a 0.678	2.45e34	equivale a 2.45×10^{34}	.56e-12	equivale a 0.56×10^{-12}
Tipo di dato	Class																		
Intero	int																		
Reale in virgola mobile	float																		
Complesso	complex (parte reale e parte immaginaria)																		
Booleano	boolean (assume solo i valori True o False)																		
Stringa	str																		
.678	equivale a 0.678																		
2.45e34	equivale a 2.45×10^{34}																		
.56e-12	equivale a 0.56×10^{-12}																		

```

Python 3.9.0 Shell
File Edit Shell Debug Options Window Help
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct 5 2020, 15:23:07) [MSC v.1927 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> a=45
>>> type(a)
<class 'int'>
>>> b=23.4
>>> b
23.4
>>> type(b)
<class 'float'>
>>> c=.85e-3
>>> c
0.00085
>>> type(c)
<class 'float'>
>>> d=a<b
>>> d
False
>>> type(d)
<class 'bool'>
>>> x=3+4j
>>> x
(3+4j)
>>> type(x)
<class 'complex'>
>>> s="computer"
>>> type(s)
<class 'str'>
>>> q='c'
>>> type(q)
<class 'str'>
>>> |

```

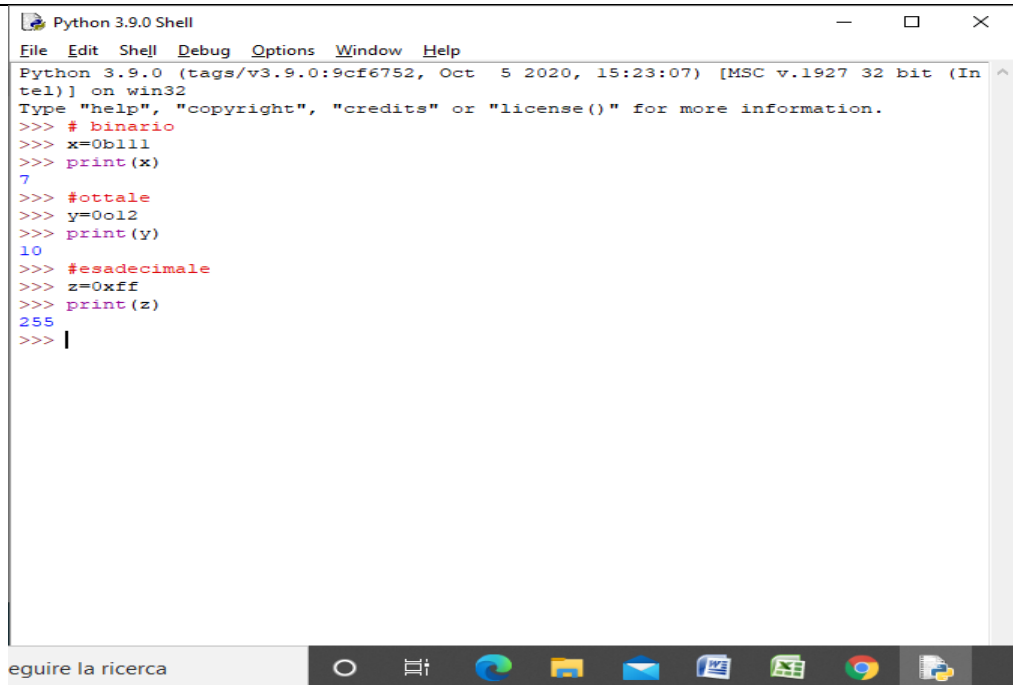
N.B.:

anche il singolo carattere è considerato di classe *str*.

Sistemi di numerazione diversa dal decimale

Si possono rappresentare anche i numeri in **base diversa da 10** con i seguenti prefissi, formati dalla cifra zero seguita da una lettera minuscola o maiuscola:

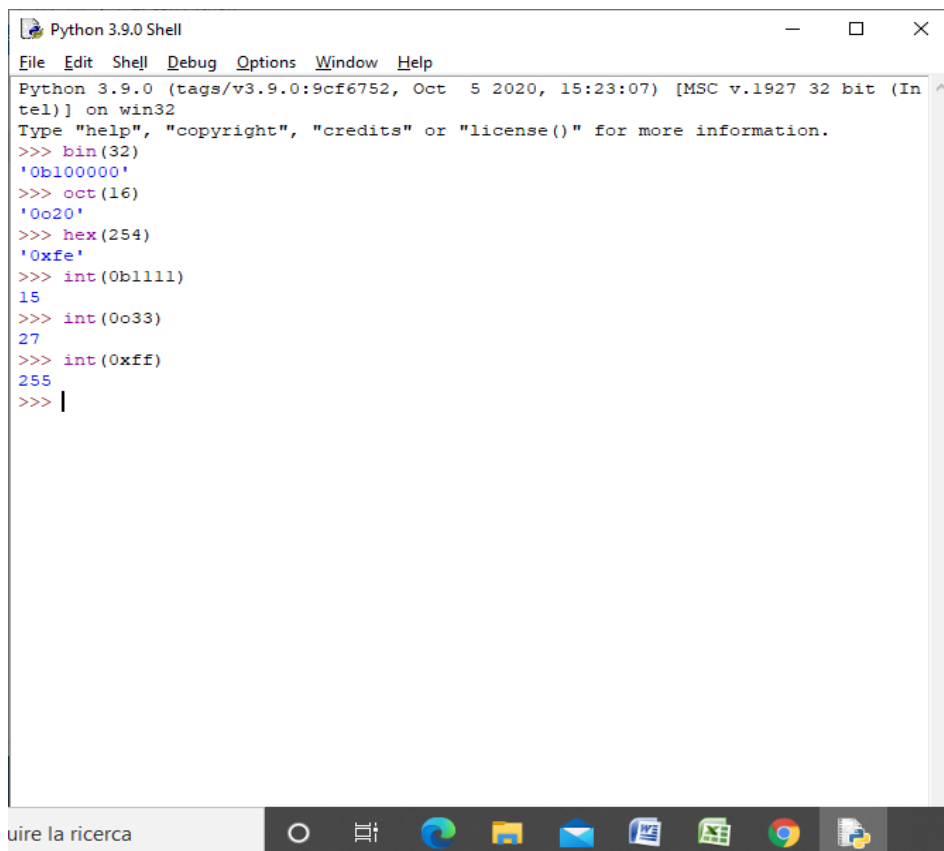
Base	Sistema di numerazione	Prefisso
2	binario	0b 0B
8	ottale	0o 0O
16	esadecimale	0x 0X



```
Python 3.9.0 Shell
File Edit Shell Debug Options Window Help
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct 5 2020, 15:23:07) [MSC v.1927 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> # binario
>>> x=0b111
>>> print(x)
7
>>> #ottale
>>> y=0o12
>>> print(y)
10
>>> #esadecimale
>>> z=0xff
>>> print(z)
255
>>> |
```

Sono disponibili le funzioni predefinite per il passaggio da un sistema ad un altro:

- `bin()` per convertire un intero in binario;
- `oct()` per convertire un intero in ottale;
- `hex()` per convertire un intero in esadecimale;
- `int()` per convertire in decimale.



```
Python 3.9.0 Shell
File Edit Shell Debug Options Window Help
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct 5 2020, 15:23:07) [MSC v.1927 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> bin(32)
'0b100000'
>>> oct(16)
'0o20'
>>> hex(254)
'0xfe'
>>> int(0b1111)
15
>>> int(0o33)
27
>>> int(0xff)
255
>>> |
```

Osservazione:

- il valore restituito dalla funzione `input()`, per acquisire i dati da *standard input* (*tastiera*), è di tipo stringa; se si vuole utilizzare il valore inserito in calcoli

numerici, occorre convertire la stringa in intero o reale con le funzioni predefinite **int()** oppure **float()** (conversione esplicita o **casting**).

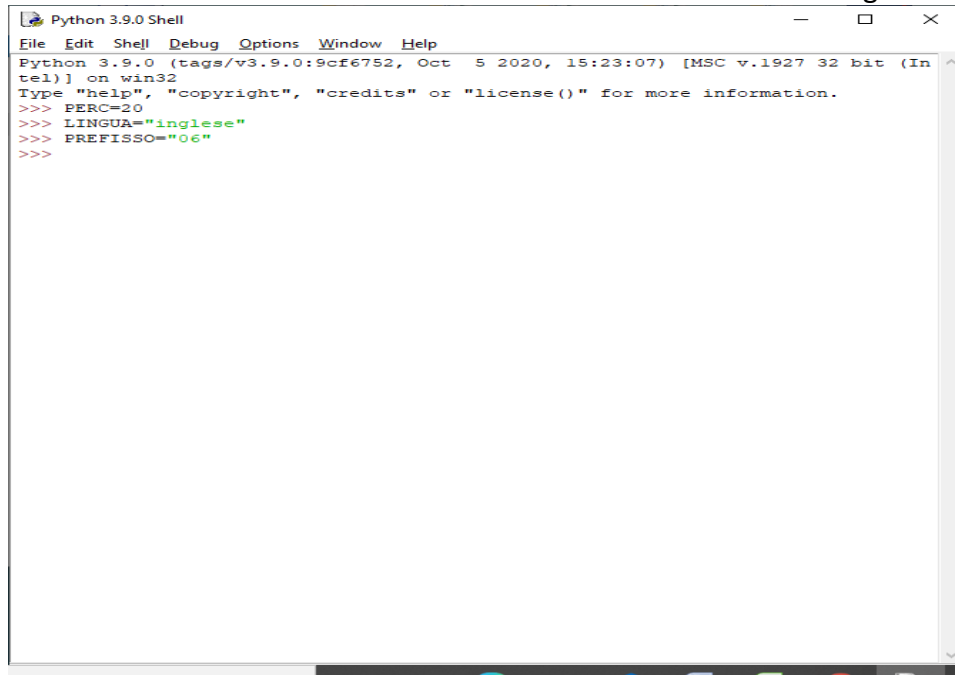
- Nelle espressioni di calcolo con operandi di diverso tipo, i numeri interi sono automaticamente convertiti in numeri float (conversione implicita o **coercizione**).

Le costanti

Le costanti differiscono dalle variabili perché rappresentano valori che non cambiano al passare del tempo.

Le costanti sono rappresentate con mnemonici aventi lettere tutte minuscole.

Il valore di inizializzazione di una costante è scritto a destra del segno dell'uguale.

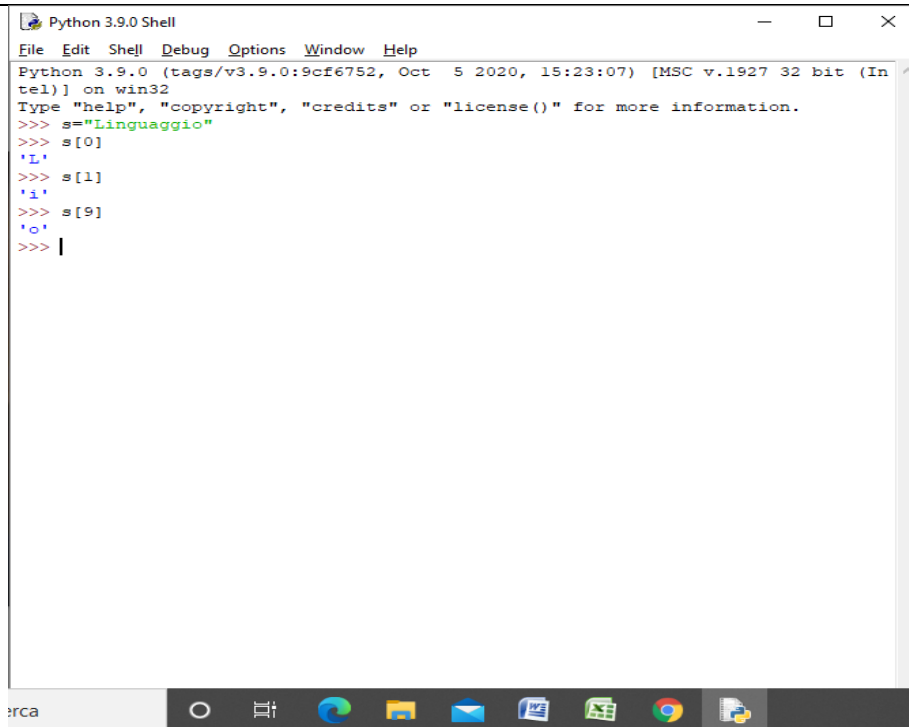
A screenshot of a Python 3.9.0 Shell window. The window title is "Python 3.9.0 Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The status bar at the bottom shows "Python 3.9.0 (tags/v3.9.0:9cf6752, Oct 5 2020, 15:23:07) [MSC v.1927 32 bit (Intel)] on win32". The main text area contains the following text:

```
Type "help", "copyright", "credits" or "license()" for more information.
>>> PERC=20
>>> LINGUA="inglese"
>>> PREFISSO="06"
>>>
```

Le stringhe come sequenze di carattere

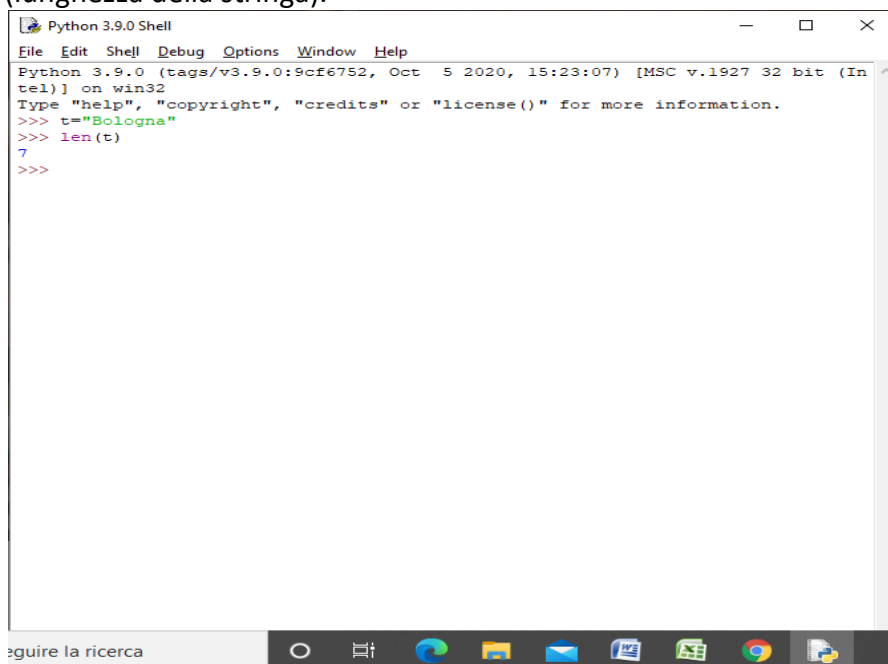
Le stringhe possono essere considerate come sequenze di caratteri, a ciascuno dei quali si associa un **indice** che inizia da 0.

Per estrarre un singolo carattere da una stringa, si scrive il nome della stringa con l'indice tra **parentesi quadre**.



```
Python 3.9.0 Shell
File Edit Shell Debug Options Window Help
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct 5 2020, 15:23:07) [MSC v.1927 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> s="Linguaggio"
>>> s[0]
'L'
>>> s[1]
'i'
>>> s[9]
'o'
>>> |
```

La funzione predefinita **len()** restituisce il numero di caratteri presenti in una stringa (lunghezza della stringa).



```
Python 3.9.0 Shell
File Edit Shell Debug Options Window Help
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct 5 2020, 15:23:07) [MSC v.1927 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> t="Bologna"
>>> len(t)
7
>>>
```

n.b.:

L'indice dei caratteri può assumere valori compresi tra 0 e $len(t)-1$.

Un singolo carattere è una stringa di lunghezza 1.

L'indicizzazione dei caratteri può essere considerata a partire **da destra**: in questo caso l'indice è negativo. L'ultimo carattere a destra ha indice -1 ; poi proseguendo verso sinistra, gli indici sono -2 , -3 , etc.

Es.:

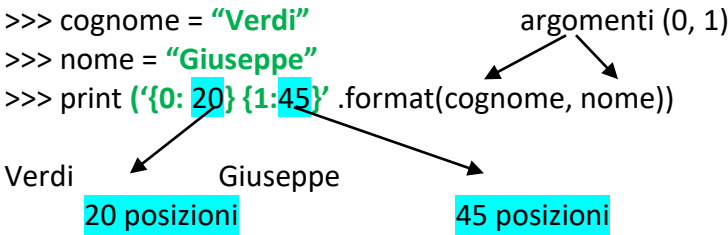
```
>>> w = "Palermo"
```

	<pre>>>> w[-1] 'o' >>> w[-2] 'm' >>> w[-7] 'p'</pre> <p>n.b.:</p> <p>Se si usa come indice un valore superiore alla lunghezza della stringa si ottiene un messaggio di errore.</p> <p>Es.</p> <pre>>>> w[12] IndexError: string index out of range >>></pre>
Slicing	<p>L'indicizzazione dei caratteri di una stringa consente di estrarre le sottostringhe, indicando tra parentesi quadre due numeri separati dai due punti (:).</p> <p>Vengono estratti i caratteri dal primo indice al secondo indice escluso. Se si indica solo il primo indice, la sottostringa viene estratta dall'indice fino alla fine della stringa. Se si indica solo il secondo indice, la sottostringa viene estratta a partire dall'inizio fino al secondo indice escluso.</p> <p>Es.:</p> <pre>>>> s = "automobile" >>> s[0:4] 'auto' >>> s[4:8] 'mobi' >>> s[:4] 'auto' >>> s[4:] 'mobile'</pre> <p>Quindi l'intera stringa può essere ottenuta come concatenazione (operatore +) di due parti, indicando lo stesso indice prima dei due punti e dopo i due punti:</p> $t = s[:i] + s[i:]$ <p>Esempio:</p> <pre>>>> t = s[:4] + s[4:] >>> t 'automobile'</pre> <p>n.b.:</p> <p>Sulle sottostringhe NON si possono fare operazioni di assegnamento, perché le stringhe sono considerate oggetti non modificabili.</p> <p>L'istruzione di assegnamento di un carattere alla sottostringa provoca la visualizzazione di un messaggio di errore.</p> <p>Esempio:</p>

	<pre>>>> s[3] = "x"</pre> <p>TypeError: 'str' object does not support item assignment</p>
Lo Username Definire lo username per l'utente di un servizio Web	<p>Lo username è costruito concatenando le prime tre lettere del cognome, le prime tre lettere del nome e le ultime due cifre della data di nascita.</p> <p>Si vuole anche che lo username sia scritto con le lettere tutte minuscole. Per trasformare una stringa in caratteri tutti minuscoli, si usa la notazione lower:</p> <p style="text-align: center;">s.lower()</p> <p>in modo analogo, la notazione upper consente di ottenere i caratteri tutti maiuscoli:</p> <p style="text-align: center;">s.upper</p> <p>n.b. <i>lower</i> ed <i>upper</i> sono metodi dell'oggetto s di classe str.</p> <p>Poiché non è possibile prevedere a priori in quale formato l'utente inserirà da tastiera la propria data di nascita (<numero di cifre per giorno, mese e anno, con o senza barre di separazione etc.), conviene estrarre i caratteri dalla data di nascita da destra, usando gli indici negativi.</p> <p>La seguente notazione estrae dalla stringa data nascita i caratteri dal penultimo (indice = -2) fino alla fine della stringa:</p> <p>data nascita[-2:]</p>
Coding	<p># username.py: nome dell'utente di un servizio Web</p> <p># input dei dati</p> <pre>cognome = input("Cognome: ") nome = input("Nome: ") datanascita = input("Data di nascita (gg/mm/aa): ") # costruzione dello username uname = cognome[:3] + nome[:3] + data_nascita[-2:] uname = uname.lower() # output del risultato Print ("Il tuo username è: ", uname)</pre> <p>n.b. dopo la visualizzazione del risultato, la finestra si chiude. Per lasciare la finestra aperta e poter controllare i risultati, si può inserire alla fine del programma una istruzione fittizia di input che mantiene aperta la finestra fino alla pressione di un tasto qualsiasi:</p> <p style="text-align: center;">tasto= input()</p>
Le opzioni di print	<p>La funzione print scrive sul video (<i>standard output</i>) i messaggi o i valori delle variabili, indicati come argomenti e separati tra loro con la virgola.</p> <p>Esempio</p> <pre>>>> p = 1200 >>> print ("Il prezzo finale è", p)</pre> <p>Il prezzo finale è 1200</p>

	<p>Dopo la visualizzazione, il cursore ritorna automaticamente a capo. Se si vuole inserire una riga vuota, si usa la sequenza <code>\n</code> (<i>newline</i>).</p> <p>Esempio</p> <pre>>>> print (" Il prezzo finale è", p, "\n al netto dell'IVA") Il prezzo finale è 1200 Al netto dell'IVA</pre>
I metacaratteri	<p>I caratteri come la barra rovesciata <code>\</code>, le doppie virgolette <code>"</code> e l'apostrofo <code>'</code>, quando sono usati come elementi della sintassi del linguaggio, si chiamano metacaratteri.</p> <p>Per togliere il significato ai metacaratteri in una stringa, basta inserire prima una barra rovesciata.</p> <p>Esempio</p> <p>Il seguente percorso di un file su disco è visualizzato corrtamente:</p> <pre>>>> path = "C:\esercizi\ese1.py" >>> print (path) C:\esercizi\ese1.py</pre> <p>Se invece il nome del file inizia con la lettere n, Python interpreta la sequenza <code>\n</code> come un ritorno a capo:</p> <pre>>>> path = "C:\esercizi\nazioni.py" >>> print (path) C:\esercizi nazioni.py</pre> <p>Quindi si utilizzano due backslash per rappresentare una barra rovesciata, togliendole il significato di metacarattere:</p> <pre>>>> path = "C:\\esercizi\\nazioni.py" >>> print (path) C:\esercizi\nazioni.py</pre> <p>In alternativa si può utilizzare una raw string (letteralmente "stringa grezza"), facendo precedere la lettera r alla stringa da visualizzare:</p> <pre>>>>print (r"C:\esercizi\nazioni.py") C:\esercizi\nazioni.py</pre> <p>L'uso della barra rovesciata <code>\</code> consente di visualizzare un metacarattere qualsiasi, come il carattere <i>virgolette</i> che, nella sintassi del linguaggio, è delimitatore delle stringhe.</p> <p>Esempio</p> <pre>>>> print (" Il titolo del libro è \"Informatica\"") Il titolo del libro è "Informatica"</pre> <p>Le triple virgolette</p> <p>Le triple virgolette (oppure i triplici apici)delimitano una stringa che viene visualizzata così come è scritta, compresi spazi e ritorni a capo.</p> <p>Esempio</p> <p>Il seguente codice è la parte iniziale del programma per il calcolo dell'area del</p>

	<div>triangolo:</div> <div># triangolo.py: area del triangolo</div> <div>print ("""<div>il carattere\ indica che l'istruzione continua nella riga successiva,</div><div>quindi non viene considerato il ritorno a capo</div>questo programma calcola l'area del triangolo conoscendo le misure di base ed altezza. La formula è: b x h / 2 """)</div> <div># input dei dati</div> <div>b = int(input("Misura della base: "))</div> <div>.....</div>																																				
Le sequenze di escape	<div>La sequenza di caratteri \n, vista a pag. 17 per specificare un salto di riga (newline), si chiama sequenza di escape.</div> <div>La tabella presenta un elenco delle sequenze di escape utilizzate dal linguaggio Python:</div> <table><thead><tr><th>Sequenza di escape</th><th>Descrizione</th><th>Terminologia inglese</th></tr></thead><tbody><tr><td>\n</td><td>A capo di una riga</td><td>newline</td></tr><tr><td>\t</td><td>Tabulazione</td><td>tab</td></tr><tr><td>\r</td><td>Ritorno a capo della stessa riga</td><td>carriage return</td></tr><tr><td>\"</td><td>Doppi apici</td><td>double quote</td></tr><tr><td>\\</td><td>Barra rovesciata</td><td>backslash</td></tr><tr><td>\b</td><td>Una battuta indietro</td><td>backspace</td></tr><tr><td>\'</td><td>Apice singolo</td><td>single quote</td></tr><tr><td>\?</td><td>Punto di domanda</td><td>question mark</td></tr><tr><td>\a</td><td>Segnalazione acustica</td><td>bell</td></tr><tr><td>\f</td><td>Salto di pagina</td><td>form feed</td></tr><tr><td>\u</td><td>Carattere unicode</td><td>unicode</td></tr></tbody></table>	Sequenza di escape	Descrizione	Terminologia inglese	\n	A capo di una riga	newline	\t	Tabulazione	tab	\r	Ritorno a capo della stessa riga	carriage return	\"	Doppi apici	double quote	\\	Barra rovesciata	backslash	\b	Una battuta indietro	backspace	\'	Apice singolo	single quote	\?	Punto di domanda	question mark	\a	Segnalazione acustica	bell	\f	Salto di pagina	form feed	\u	Carattere unicode	unicode
Sequenza di escape	Descrizione	Terminologia inglese																																			
\n	A capo di una riga	newline																																			
\t	Tabulazione	tab																																			
\r	Ritorno a capo della stessa riga	carriage return																																			
\"	Doppi apici	double quote																																			
\\	Barra rovesciata	backslash																																			
\b	Una battuta indietro	backspace																																			
\'	Apice singolo	single quote																																			
\?	Punto di domanda	question mark																																			
\a	Segnalazione acustica	bell																																			
\f	Salto di pagina	form feed																																			
\u	Carattere unicode	unicode																																			
L'output formattato	<div>I risultati di un'elaborazione o i messaggi per l'utente possono essere visualizzati in modo leggibile ed ordinato usando la formattazione dell'output, grazie alla quale dati numerici e stringhe sono presentati secondo un formato controllato dal programmatore.</div> <div>Il formato che viene definito con una stringa, delimitata da apici o da virgolette, seguita dal punto e dalla parola chiave format().</div> <div>n.b.:</div> <div>format() è un metodo applicato ad un oggetto di tipo stringa (str).</div> <div>All'interno della stringa vengono inseriti i segnaposto (placeholder) racchiusi tra una coppia di parentesi graffe {...}, che specificano il formato del dato da visualizzare, mentre format() ha come argomento, tra parentesi tonde, il dato da visualizzare.</div> <div>Per visualizzare un numero in formato float all'interno di 10 posizioni, con Cifre decimali, si scrive la seguente istruzione print:</div> <div>>>> p = 1200</div>																																				

	<pre>>>> print(' Il prezzo finale è {: 10.2f}' . format(p))</pre> <p>Il prezzo finale è 1200.00</p> <p>Come si può notare dall'esempio precedente, il segnaposto contiene, dopo la parentesi graffa, il carattere: (due punti), il numero di posizioni, il punto, il numero di cifre decimali, la lettera <i>f</i> per i numeri <i>float</i>. Inoltre <i>format()</i> ha come argomento la variabile <i>p</i> da stampare.</p> <p>Le specifiche di formato sono contrassegnate con le lettere:</p> <ul style="list-style-type: none"> • d per i numeri interi; • f per i numeri reali; • s per le stringhe. <p>Esempi</p> <pre>>>> età = 18 >>> print ('La persona ha {: 3d} anni'.format (età))</pre> <p>La persona ha 18 anni</p> <pre>>>> cognome = "Rossi" >>> print ('{: 12s}'.format(cognome))</pre> <p>Rossi</p> <p>n.b. nel caso delle stringhe, la specifica <i>s</i> può essere omessa.</p>
<p>Gli indici posizionali</p>	<p>Frequentemente la visualizzazione riguarda più dati: in questi casi occorre usare più segnaposti che vengono associati ai dati mediante un indice che parte da 0, scritto nel formato prima del carattere:</p> <p>l'indice è posizionale, nel senso che il formato con indice 0 è associato al primo dato specificato negli argomenti di <i>format()</i>, l'indice 1 al secondo e così via.</p> <p>Esempio Per le stringhe:</p> <pre>>>> cognome = "Verdi" >>> nome = "Giuseppe" >>> print ('{:0: 20} {1:45}'.format(cognome, nome))</pre>  <p>Verdi Giuseppe</p> <p>20 posizioni 45 posizioni</p> <p>Per i dati numerici di tipo intero (specifica <i>d</i>)</p> <p>Esempio</p> <pre>>>> a = 10 >>> print('{0:3d} {1:5d} {2:10d}'.format(a, a*2, a*3))</pre> <p>10 20 30</p> <p>Per i dati numerici di tipo <i>float</i> (specifica <i>f</i>)</p> <pre>>>> peso = 0.160</pre>

	<pre>>>> prezzo = 6.30 >>> totale = peso * prezzo >>> print ('Peso = {0: 5.3f} Prezzo = {1:10.2f} Totale = {2:10.2f}'. format(peso, prezzo, totale)) peso = 0.160 prezzo = 6.30 totale = 1.01</pre> <p>Con le unità di misura:</p> <pre>>>> print ('peso = {0:5.3f} kg prezzo = {1:10.2f} euro/kg totale = {2:10.2f}euro'.format(peso, prezzo, totale)) Peso = 0.160 kg prezzo = 6.30 euro/kg totale = 1.01 euro</pre> <p>Con il simbolo dell'euro (carattere Unicode 20AC):</p> <pre>>>> print('peso = {0:5.3f} kg prezzo = {1:10.2f} \u20AC/kg totale = {2:10.2f} \u20AC'.format(peso, prezzo, totale)) Peso = 0.160 kg prezzo = 6.30 €/kg totale = 1.01 €</pre> <p>Nei dati di tipo <i>float</i>, l'impostazione del numero di cifre decimali ne comporta l'arrotondamento.</p>
L'allineamento	<p>Per default, le stringhe sono allineate a sinistra e i dati numerici sono allineati a destra. Questa impostazione può essere modificata inserendo, dopo il carattere : uno di questi segni:</p> <ul style="list-style-type: none"> • < per l'allineamento a sinistra (se non è già impostato di default) • ^ per l'allineamento centrato (sia stringhe che numeri) • > per l'allineamento a destra (se non è già impostato di default) <p>Esempio</p> <pre>>>> print(' {0:>20} {1:>45}'.format(cognome, nome)) Verdi Giuseppe >>> print(' {0:^20} {1:^45}'.format(cognome, nome)) Verdi Giuseppe</pre>
I segni	<p>I dati di tipo numerico possono essere visualizzati con il segno. In particolare il simbolo +, dopo i <i>due punti</i>, visualizza i numeri positivi preceduti dal segno + e i numeri negativi preceduti dal segno -; il simbolo spazio, dopo i <i>due punti</i>, visualizza il segno solo per i numeri negativi.</p> <p>Esempio</p> <pre>>>> x = 12 >>> y = -18 >>> print ('{0:+d} {1:+d}'.format(x, y)) +12 -18 >>> print ('{0: d} {1: d}'.format(x, y)) non visualizza il segno + ma lascia lo 12 -18 spazio</pre>
Le stringhe formattate	<p>In alternativa all'uso del metodo <i>format()</i>, si possono costruire stringhe formattate delimitate da apici (o da virgolette) e precedute dalla lettera f, dette <i>formatted string literals</i> o, in modo abbreviato, f-strings.</p>

	<p>Il dato da visualizzare è indicato direttamente nella stringa formattata all'interno del segnaposto con le parentesi graffe.</p> <p>Esempio Considerando i dati: <pre>>>> a, b, c = 10, 20, 30</pre></p> <p>L'istruzione <pre>>>> print('{0:3d} {1:5d} {2:10d}'.format(a, b, c))</pre> <pre>10 20 30</pre></p> <p>È equivalente alla seguente: <pre>>>> print(f'{a:3d} {b:5d} {c:10d}')</pre> <pre>10 20 30</pre></p> <p>Considerando i dati: <pre>>>> città = "Roma" >>> regione = "Lazio"</pre></p> <p>L'istruzione: <pre>>>> print('{0:25} si trova in {1:20}'.format(città, regione))</pre> <pre>Roma si trova in Lazio</pre></p> <p>Produce lo stesso risultato della seguente: <pre>>>> print(f'{città:25} si trova in {regione: 20}')</pre> <pre>Roma si trova in Lazio</pre></p> <p>Considerando i dati: <pre>>>> lato = 3 >>> area = lato ** 2</pre></p> <p>L'istruzione: <pre>>>> print('L\'area del quadrato di lato {0:3d} è = {1:5d}'.format(lato, area))</pre> <pre>L'area del quadrato di lato 3 è = 9</pre></p> <p>Produce lo stesso risultato della seguente: <pre>>>> print(f'L\'area del quadrato di lato {0:3d} è = {area:5d}')</pre> <pre>L'area del quadrato di lato 3 è = 9</pre></p> <p>n.b.: nell'esempio la notazione <code>\'</code> toglie il significato al metacarattere apice.</p> <p>Le stringhe formattate si distinguono dalle normali stringhe perché contengono campi di segnaposto (delimitati dalle parentesi graffe), che vengono sostituiti in fase di esecuzione (runtime).</p>
Coding	<p>Il cambio EURO/DOLLARI</p> <p>Scrivere un programma che richieda all'utente l'importo in euro da cambiare ed il cambio euro/dollaro e che comunichi il corrispondente valori in dollari.</p> <p>Il programma seguente mostra un esempio di utilizzo degli output formattati con l'allineamento centrato così da visualizzare in modo più ordinato i risultati dell'elaborazione.</p>

	<p>Le variabili sono di tipo <i>float</i>, in quanto sia i dollari che gli euro sono numeri con i decimali ed il valore del cambio può avere una parte decimale.)</p> <pre> # cambio.py : cambio di denaro da euro a dollari print("""\ programma per calcolare il valore di una somma in dollari conoscendo il cambio euro/dollaro """) #dati di input euro = float(input("Importo in euro? ")) cambio = float(input("Cambio euro/dollaro? ")) # calcolo dollari= cambio * euro # output del risultato print('{0:^20} {1:^20} {2:^20}'.format("Euro", "Cambio", "Dollari")) print('{0:^20.2f} {1:^20.2f} {2:^20.2f}'.format(euro, cambio, dollari)) Esecuzione ==== RESTART: C:\Users\Studente\Desktop\ex con Python\cambio_euro_dollari.py ==== programma per calcolare il valore di una somma in dollari conoscendo il cambio euro/dollaro Importo in euro? 2000 Cambio euro/dollaro? 1.13 Euro Cambio Dollari 2000.00 1.13 2260.00 </pre>
Le Liste	<p>Una lista è un'aggregazione di dati racchiusi tra una coppia di parentesi quadre e separati da virgola. E' quindi un dato composto, cioè non elementare.</p> <p>Gli elementi della lista possono essere di tipo diverso (numeri e stringhe), anche se nella maggior parte delle applicazioni le liste sono usate per rappresentare insieme di dati omogenei tra loro.</p> <p>Esempio</p>

```
Python 3.9.0 Shell
File Edit Shell Debug Options Window Help
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct 5 2020, 15:23:07) [MSC v.1927 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> dati= ['Angela', 51, 'Maria', 12, 'Dario', 34, 'Marco', 52]
SyntaxError: invalid syntax
>>> dati= ['Angela', 51, 'Maria', 12, 'Dario', 34, 'Marco', 52]
>>> dati
['Angela', 51, 'Maria', 12, 'Dario', 34, 'Marco', 52]
>>> targhe = ['AB345CV', 'FR678AM', 'ER772GD']
>>> targhe
['AB345CV', 'FR678AM', 'ER772GD']
>>> numeri= [10, 20, 30, 40]
>>> print(numeri)
[10, 20, 30, 40]
>>> |
```

La lista è un insieme ordinato di dati, nel senso che ogni elemento della lista è associato ad un indice che inizia da 0.

Esempio

```
>>> print(<dati[0], dati[1])
Angela 51
>>> print( targhe [2])
ER772GD
>>> print (numeri[1])
20
```

In analogia a quanto accade con le stringhe, si possono **estrarre sottoliste** delle liste (*slicing*), usando la notazione con i due punti:

indice1:indice2	da indice1 a indice2 escluso;
indice1	da indice1 alla fine della lista;
:indice2	dall’inizio della lista fino a indice2 escluso

Esempio

```
>>> dati[2:5]
['Maria', 12, 'Dario']
dati[2:]
['Maria', 12, 'Dario', 34, 'Marco', 52]
>>> dati[:4]
['Angela', 51, 'Maria', 12]
>>>
```

n.b.:

	<p>a differenza delle stringhe, gli elementi delle liste possono essere modificati, assegnando un nuovo valore all'elemento identificato attraverso l'indice</p> <p>Esempio</p> <pre>numeri [10, 20, 30, 40] >>> numeri[3]= 55 >>> numeri [10, 20, 30, 55] >>></pre> <p>Nella terminologia del linguaggio Python si dice che la stringa è un esempio di oggetto immutabile e la lista è un esempio di oggetto mutabile.</p>
La funzione len()	<p>La funzione predefinita len() restituisce il numero degli elementi della lista.</p> <p>Esempio</p> <pre>>>> len(dati) 8</pre>
Le strutture di controllo	<p>Teorema di Bohm- Jacopini (1966)</p> <p>Un qualsiasi algoritmo può essere espresso usando esclusivamente le strutture di sequenza, selezione e di iterazione.</p>
Coding	<p>Es. 1</p> <p>Calcolare la differenza tra due numeri.</p> <p>Programma Python (differenza.py)</p> <pre># differenza.py: operazione di sottrazione # input dei dati num1 = int(input("Primo numero: ")) num2 = int(input("Secondo numero: ")) # calcolo differenza = num1 - num2 # output del risultato print("Risultato = ", differenza)</pre> <p>Es.2</p> <p>Scambiare il contenuto di due variabili.</p> <p>Programma Python (swap.py)</p> <pre># scambio.py: scambio del valore di variabili # dati x = 1 y = 1 print("prima:", x, y) # swap</pre>

```
temp = x
x = y
y = temp
# variabili scambiate
print("dopo:", x, y)
```

Es. 3

Calcolare lo sconto del 20% sul prezzo di un articolo.

Programma Python (calcola Sconto.py)

```
# CalcoloSconto.py: calcolo del prezzo scontato
# input dei dati
descrizione = input("Descrizione del prodotto: ")
prezzo = float(input("Prezzo: "))
# calcoli
sconto = prezzo * 20 / 100
prezzo = prezzo - sconto
# output del risultato
Print('Prezzo scontato di {0:20s} = {1:10.2f}'.format(descrizione, prezzo))
```

Es. 4

Calcolare il perimetro e l'area di un triangolo rettangolo dopo aver acquisito da tastiera le misure dei due cateti.

Possiamo utilizzare le **funzioni matematiche predefinite** del linguaggio:
math.sqrt(x) per il calcolo della radice quadrata (square root) di x
math.pow(x, y) per il calcolo della potenza x^y

Per rendere disponibili queste funzioni occorre inserire all'inizio del programma il comando **import** di importazione della **libreria math** di Python

Programma Python (triangolo Rettangolo.py)

```
# TriangoloRettangolo.py: perimetro ed area di un triangolo rettangolo
# importazione della libreria delle funzioni matematiche
```

```
import math
```

```
# input dei dati
cateto1 = float(input("Primo cateto"))
cateto2 = float(input("Secondo cateto"))
# calcoli
ipotenusa = math.sqrt(math.pow(cateto1, 2) + math.pow(cateto2, 2))
perimetro = cateto1 + cateto2 + ipotenusa
area = cateto1 * cateto2 / 2
# output dei risultati
print('Perimetro = {:.3f}'.format(perimetro))
print('Area = {:.3f}'.format(area))
```

	<p>n.b.: La funzione predefinita <i>math.pow</i> effettua una conversione implicita al tipo float sia per la base che per l'esponente. Se si desidera ottenere la potenza intera di un numero intero si deve usare l'operatore <i>**</i>.</p>																
I connettivi logici	<p>In un programma si può avere bisogno di ricorrere agli operatori booleani (o connettivi logici) and, or, not.</p> <table><tr><th>Operazione</th><th>Operatore</th><th>Significato</th><th>Esempio</th></tr><tr><td>Congiunzione</td><td>and</td><td>Il risultato è true se entrambi gli operandi sono true; sarà false negli altri casi.</td><td>A and B</td></tr><tr><td>Disgiunzione</td><td>or</td><td>Il risultato è true se almeno uno dei operandi è true; sarà false se sono entrambi false.</td><td>A or B</td></tr><tr><td>Negazione</td><td>not</td><td>Il risultato è true se l'operando è false; il risultato è false se l'operando è true.</td><td>not B</td></tr></table> <p>n.b. si può utilizzare anche una scrittura del tipo: a <= x <= b per rappresentare il fatto che il valore di x è compreso tra il valore di a ed il valore di b. l'espressione precedente equivale a: x >= a and x <= b</p> <p>Esempio >>> a = 1 >>> b = 5 >>> x = 3 >>> a <= x <= b True</p> <p>Il valore di queste espressioni può essere True o False e può essere assegnato a variabili di tipo boolean.</p> <p>Esempio >>> x = a == 3 or b == 5 >>> print(x) True</p>	Operazione	Operatore	Significato	Esempio	Congiunzione	and	Il risultato è true se entrambi gli operandi sono true; sarà false negli altri casi.	A and B	Disgiunzione	or	Il risultato è true se almeno uno dei operandi è true; sarà false se sono entrambi false.	A or B	Negazione	not	Il risultato è true se l'operando è false; il risultato è false se l'operando è true.	not B
Operazione	Operatore	Significato	Esempio														
Congiunzione	and	Il risultato è true se entrambi gli operandi sono true; sarà false negli altri casi.	A and B														
Disgiunzione	or	Il risultato è true se almeno uno dei operandi è true; sarà false se sono entrambi false.	A or B														
Negazione	not	Il risultato è true se l'operando è false; il risultato è false se l'operando è true.	not B														
La selezione	<p>Per la selezione (o struttura di alternativa) nel linguaggio Python si usa l'istruzione if.....else..... che ha la seguente sintassi: if condizione: blocco1 else: blocco2</p>																
Coding	Dati in input due numeri, disporli in ordine crescente.																

I: a,b
O: a,b oppure b,a

Inizio(elabora)

Chiedo, scrivo a, b

Se $a < b$ allora

Visualizza a,b

Altrimenti

Visualizza b,a

fine se

fine

ordina.py: due numeri in ordine crescente

input dei dati

```
a = int(input("primo numero: "))
```

```
b = int(input("secondo numero: "))
```

#numeri ordinati

```
if a < b:
```

```
    print(a)
```

```
    print(b)
```

```
else:
```

```
    print(b)
```

```
    print(a)
```

Gli operatori nella condizione

La condizione è un'espressione booleana di cui viene valutata la verità: vengono quindi utilizzati gli operatori di confronto `==`, `<`, `>`, `<=`, `>=`, `!=` e gli operatori booleani `and`, `or`, `not` per costruire espressioni logiche combinando tra loro più condizioni.

Esempi

Il seguente frammento di programma produce in output il nome di uno studente solo nel caso in cui si verifichino entrambe le condizioni:

```
if classe == 5 and anni > 18:
```

```
    print(nome)
```

Nel frammento di programma di seguito elencato, la struttura di selezione produce in output il valore di numero nel caso in cui si verifichino una o entrambe le condizioni:

```
if numero % 2 == 0 or numero > 50:
```

```
    print(numero)
```

La radice quadrata

Calcolare la radice quadrata di un numero.

Il calcolo viene effettuato solo se il radicando è ≥ 0 .

Il programma utilizza la funzione predefinita **math.sqrt** per il calcolo della radice quadrata. All'inizio del programma occorre importare la libreria `math` per le funzioni matematiche.

	<pre> Programma Python (radice.py) # radice.py: radice quadrata di un numero import math # input del radicando n = float(input("numero: ")) # controlla che sia maggiore o uguale a zero if n >= 0: print('{:10.5f}'.format(math.sqrt(n))) else: print("Il numero inserito è negativo") </pre>										
La selezione multipla	<p>Il linguaggio Python non possiede una specifica istruzione per rappresentare la struttura di selezione multipla, come le istruzioni switch o case di altri linguaggi di programmazione.</p> <p>La selezione multipla è considerata una <i>struttura derivata dalla</i> struttura fondamentale di selezione binaria ed è costruita utilizzando una successione di selezione annidate.</p> <p>La selezione annidata è rappresentata con la parola chiave elif (abbreviazione di <i>else if</i>), secondo il seguente schema:</p> <pre> if condizione1: blocco1 elif condizione2: blocco2 elif condizionen: bloccon else: blocco </pre>										
Coding	<p>Lo sconto progressivo Per la vendita di un prodotto, applicare uno sconto progressivo in base al numero di pezzi ordinati.</p> <p>Lo sconto viene applicato secondo la seguente tabella:</p> <table border="1"> <thead> <tr> <th>Pezzi</th><th>Sconto</th></tr> </thead> <tbody> <tr> <td>Fino a 3</td><td>5%</td></tr> <tr> <td>Fino a 5</td><td>10%</td></tr> <tr> <td>Fino a 10</td><td>20%</td></tr> <tr> <td>Più di 10</td><td>30%</td></tr> </tbody> </table> <p>I: pezzi, prezzo O: importo</p> <p>Programma Python (ScontoProgressivo.py) # ScontoProgressivo.py: sconto progressivo sui prodotti # input dei dati pezzi = int(input("Pezzi acquistati: ")) prezzo = float(input("Prezzo del prodotto: ")) # determina lo sconto con una selezione multipla if pezzi == 1 or pezzi == 2 or pezzi == 3:</p>	Pezzi	Sconto	Fino a 3	5%	Fino a 5	10%	Fino a 10	20%	Più di 10	30%
Pezzi	Sconto										
Fino a 3	5%										
Fino a 5	10%										
Fino a 10	20%										
Più di 10	30%										

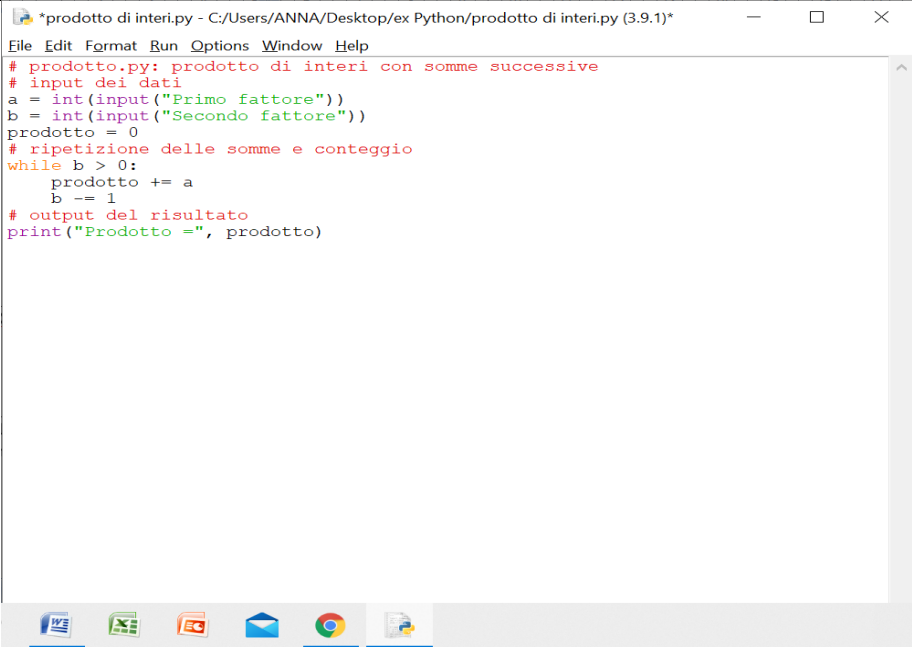
	<pre> sconto = 5 elif pezzi == 4 or pezzi == 5: sconto = 10 elif pezzi == 6 or pezzi <= 10: sconto = 20 else: sconto = 30 # calcolo e output del risultato importo = pezzi * prezzo * (100.0 - sconto)/100.0 Print ('Importo da pagare = {:12.2f} euro'.format(importo)) </pre>
La ripetizione <i>for</i>	<p>Si rappresenta in Python con l'istruzione for <i>contatore</i> in insieme:</p> <p>Ci sono due modalità per l'uso della ripetizione <i>for</i>: quando l'<i>insieme</i> è rappresentato con una lista e quando l'<i>insieme</i> è definito con un intervallo.</p> <ol style="list-style-type: none"> La ripetizione può riguardare gli elementi di una lista for variabile in lista: istruzioni il carattere : nella riga del <i>for</i> indica l'inizio di un gruppo di istruzioni indentate. Le istruzioni scritte dopo il <i>for</i> in modo indentato vengono ripetute per ogni valore che la <i>variabile</i> può assumere all'interno dei valori specificati da <i>lista</i>. Questa modalità è utile per visualizzare tutti gli elementi di una lista. <p>Esempio città = ['Venezia', 'Firenze', 'Napoli', 'Bari', 'Catania', \ 'Sassari'] # il carattere \ indica che l'istruzione continua nella riga successiva for c in città: print(c) # verranno stampati i nomi delle città contenuti nella lista</p> <ol style="list-style-type: none"> Per controllare con un contatore il numero di volte che la ripetizione viene eseguita si deve usare la funzione predefinita range(), che definisce un intervallo di valori: for variabile in range(): istruzioni la funzione <i>range()</i> può avere come argomenti: <ul style="list-style-type: none"> Il numero di valori interi, a partire da 0, usati come contatore del numero di volte che la ripetizione viene eseguita; Due numeri che indicano il valore di inizio ed il valore di fine (escluso) che il contatore assume per controllare il numero di volte che la ripetizione viene eseguita; Due numeri per l'inizio e la fine ed un terzo numero che indica l'incremento che il contatore assume ad ogni ripetizione. <p>Esempio for i in range(6): print(i)</p> <pre> 0 1 2 3 </pre>

	<pre> 4 5 for i in range(1, 6): print(i) 1 2 3 4 5 for i in range(1, 15, 3): print(i) 1 4 7 10 13 </pre>
La parola chiave end	<p>L'uso della parola chiave end, come argomento della funzione print, impedisce il ritorno a capo dopo l'output oppure indica uno specifico carattere da visualizzare al termine dell'output.</p> <p>Esempio</p> <pre> for i in range(1, 15, 3): print(i, end = ',') 1,4,7,10,13, for i in range(1, 15, 3): print(i, end = ' ') 1 4 7 10 13 </pre>
Coding	<p>La spesa annuale Calcolare il totale delle spese domestiche in un anno.</p> <p>Forniti in input gli importi (12 come i mesi dell'anno), il problema richiede di calcolare la spesa totale annuale.</p> <p>Abbiamo bisogno di un ciclo (enumerativo) che acquisisca gli importi di ciascun mese e che li sommi di volta in volta. All'uscita del ciclo viene comunicato il risultato (totale dell'anno).</p> <p>Organizziamo in una lista mesi i nomi dei mesi (prime tre lettere).</p> <p>Per ricordare all'utente il nome del mese per il quale deve inserire il valore della spesa mensile, si costruisce una stringa di messaggio <i>msg</i> che diventa l'argomento della funzione input:</p>

	<pre> msg = "Spesa mese " + m + ": " programma Python (spese.py) # spese.py: spese domestiche dell'anno mesi = ["Gen", "Feb", "Mar", "Apr", "Mag", "Giu", "Lug", \ "Ago", "Set", "Ott", "Nov", "Dic"] Totale = 0; # ripetizione sui mesi for m in mesi: msg = "Spesa mese " + m + ": " importo = float(input(msg)) totale += importo # output del risultato print('Spesa annuale: {:.12.2f}'.format(totale)) </pre>				
<p>Il prompt della funzione input()</p>	<p>Per facilitare l'input dei dati da parte dell'utente, è sempre opportuno inviare un prompt (messaggio di sollecito) che ricordi il tipo di dato da inserire. Nel Coding precedente è stata costruita una stringa di messaggio che è diventata l'argomento della funzione input().</p> <p>In alternativa si possono usare prompt formattati, in modo del tutto analogo a quanto già visto per la funzione print() nell'output formattato.</p> <p>Si possono usare, in modo equivalente, i segnaposto con le specifiche di formato oppure le stringhe formattate (<i>f-string</i>).</p> <p>Esempio</p> <p>codice = 1254 prezzo = float(input('Qual è il prezzo del prodotto {6d}? '.format(codice))) Qual è il prezzo del prodotto 1254?</p> <p>Oppure, con la f-string</p> <p>prezzo = float(input(f'Qual è il prezzo del prodotto {codice:6d}? ')) Qual è il prezzo del prodotto 1254?</p>				
<p>Coding</p>	<p>Il peso medio di n oggetti. Calcolare la media di n pesi inseriti da tastiera.</p> <table border="1" data-bbox="386 1500 1404 2024"> <tr> <td data-bbox="386 1500 775 1619"> <p>I: n, peso O: media</p> </td><td data-bbox="775 1500 1404 1619"> <p>Legenda: n = numero di oggetti da pesare</p> </td></tr> <tr> <td data-bbox="386 1619 775 2024"> <p>Inizio (calcola) somma = 0 media = 1 chiedo, scrivo n per i = 1 a n chiedo, scrivo peso somma = somma + peso i = i+1 media = somma / n visualizzo media fine</p> </td><td data-bbox="775 1619 1404 2024"> <p>Programma Python (media.py) # media.py: media di n pesi # inizializzazione della somma somma = 0 n = int(input("Quanti sono gli oggetti?: ")) for i in range(n): peso = float(input('Peso dell'oggetto n. {3d}: '.format(i+1))) somma += peso # calcolo della media dei pesi media = somma / n</p> </td></tr> </table>	<p>I: n, peso O: media</p>	<p>Legenda: n = numero di oggetti da pesare</p>	<p>Inizio (calcola) somma = 0 media = 1 chiedo, scrivo n per i = 1 a n chiedo, scrivo peso somma = somma + peso i = i+1 media = somma / n visualizzo media fine</p>	<p>Programma Python (media.py) # media.py: media di n pesi # inizializzazione della somma somma = 0 n = int(input("Quanti sono gli oggetti?: ")) for i in range(n): peso = float(input('Peso dell'oggetto n. {3d}: '.format(i+1))) somma += peso # calcolo della media dei pesi media = somma / n</p>
<p>I: n, peso O: media</p>	<p>Legenda: n = numero di oggetti da pesare</p>				
<p>Inizio (calcola) somma = 0 media = 1 chiedo, scrivo n per i = 1 a n chiedo, scrivo peso somma = somma + peso i = i+1 media = somma / n visualizzo media fine</p>	<p>Programma Python (media.py) # media.py: media di n pesi # inizializzazione della somma somma = 0 n = int(input("Quanti sono gli oggetti?: ")) for i in range(n): peso = float(input('Peso dell'oggetto n. {3d}: '.format(i+1))) somma += peso # calcolo della media dei pesi media = somma / n</p>				

		<pre>print("Peso medio = ", media)</pre> <p>esecuzione Python 3.9.0 (tags/v3.9.0:9cf6752, Oct 5 2020, 15:23:07) [MSC v.1927 32 bit (Intel)] on win32 Type "help", "copyright", "credits" or "license()" for more information. >>> ===== RESTART: C:\Users\Studiante\Desktop\ex con Python\media dei pesi.py ===== Quanti sono gli oggetti?: 5 Peso dell'oggetto n. 1: 10.6 Peso dell'oggetto n. 2: 54.6 Peso dell'oggetto n. 3: 32.8 Peso dell'oggetto n. 4: 41.9 Peso dell'oggetto n. 5: 23.6 Peso medio = 32.7</p>							
Coding	<p>Il massimo tra n numeri Data in input una serie di n numeri, determinare il numero massimo della serie.</p> <table><tr><td>I: n, dato, O: massimo</td><td>legenda: n = quanti elementi si vogliono analizzare dato = valore inserito nella sequenza</td></tr><tr><td>Algoritmo</td><td>Codifica Python</td></tr><tr><td>Inizio (calcola) Chiedo, scrivo n Per i= 1 a n Chiedo, scrivo dato Se i = 0 allora massimo = dato fine se se dato > massimo allora massimo = dato fine se i = i +1 visualizza massimo fine</td><td># massimo.py: massimo di n numeri n = int(input("Quanti sono i dati? ")) # ripetizione sui dati for i in range(n): dato = int(input(f'Inserisci il dato {i+1:2d}: ')) if i == 0: massimo = dato if dato > massimo: massimo = dato # output del valore massimo print("Il valore massimo è", massimo)</td></tr></table>			I: n, dato, O: massimo	legenda: n = quanti elementi si vogliono analizzare dato = valore inserito nella sequenza	Algoritmo	Codifica Python	Inizio (calcola) Chiedo, scrivo n Per i= 1 a n Chiedo, scrivo dato Se i = 0 allora massimo = dato fine se se dato > massimo allora massimo = dato fine se i = i +1 visualizza massimo fine	# massimo.py: massimo di n numeri n = int(input("Quanti sono i dati? ")) # ripetizione sui dati for i in range(n): dato = int(input(f'Inserisci il dato {i+1:2d}: ')) if i == 0: massimo = dato if dato > massimo: massimo = dato # output del valore massimo print("Il valore massimo è", massimo)
I: n, dato, O: massimo	legenda: n = quanti elementi si vogliono analizzare dato = valore inserito nella sequenza								
Algoritmo	Codifica Python								
Inizio (calcola) Chiedo, scrivo n Per i= 1 a n Chiedo, scrivo dato Se i = 0 allora massimo = dato fine se se dato > massimo allora massimo = dato fine se i = i +1 visualizza massimo fine	# massimo.py: massimo di n numeri n = int(input("Quanti sono i dati? ")) # ripetizione sui dati for i in range(n): dato = int(input(f'Inserisci il dato {i+1:2d}: ')) if i == 0: massimo = dato if dato > massimo: massimo = dato # output del valore massimo print("Il valore massimo è", massimo)								
Le strutture for annidate	Le strutture di controllo possono essere usate una all'interno dell'altra, in questo caso si parla di strutture annidate .								

Coding	<p>I cicli annidati Produrre sul video la tavola pitagorica.</p> <pre># TavolaPitagorica.py # ripetizione sulle righe for i in range(1, 11): # Scorre le righe # ripetizione sulle colonne for j in range(1, 11): # Scorre le colonne print(i * j, end="\t") # Usa la tabulazione come separatore tra loro i valori print() tasto= input()</pre>
Le istruzioni break e continue	<p>Con l’istruzione break (interrompi) l’esecuzione di una struttura di ripetizione (<i>while</i>-<i>for</i>) si interrompe ed il controllo passa all’istruzione successiva al blocco di istruzioni indentate.</p> <p>Esempio Nel codice del Coding per le spese domestiche (pag.31), se si vuole eseguire il programma solo per i mesi del primo semestre dell’anno, si può inserire un controllo sul nome del mese ed interrompere la ripetizione quando il mese è uguale a luglio.</p> <pre>for m in mesi: if m == "Lug": break</pre> <p>Con l’istruzione continue (<i>continua</i>), in una struttura di ripetizione (<i>while</i> o <i>for</i>), le istruzioni del blocco indentato non vengono eseguite ed il controllo ritorna all’inizio della ripetizione.</p> <p>Con riferimento all’esempio precedente, il seguente frammento di codice esegue la ripetizione per l’inserimento delle spese mensili, ma salta i mesi di luglio e agosto. Negli altri casi le istruzioni sono eseguite regolarmente.</p> <pre>for m in mesi: if m == "Lug" or m == "Ago" continue</pre>
Coding	<p>Il prodotto di interi Calcola il prodotto tra interi utilizzando la sola operazione di somma e costruire una tabella di traccia per la verifica del programma.</p>

	
<p>La gestione delle eccezioni</p>	<p>Una applicazione software ben costruita deve prevedere tutti i casi che si possono verificare durante l'esecuzione.</p> <p>Il linguaggio Python prevede la gestione delle eccezioni, intercettando le situazioni nelle quali si crea l'eccezione per evitare l'interruzione anomala del programma e inviando, di conseguenza, opportuni messaggi o avvertimenti all'utente.</p> <p>Esempio</p> <p>Si consideri la situazione che si viene a creare quando, utilizzando il programma per il calcolo della radice quadrata di un numero, l'utente inserisca come radicando un valore negativo.</p> <pre># input del radicando n = float(input("Numero: ")) # calcola la radice quadrata print('{:10.5f}'.format(math.sqrt(n)))</pre> <p>il programma dopo aver letto da tastiera il valore del radicando, si interrompe con un errore di <i>runtime</i> che produce sul video il messaggio:</p> <p>valueError: math domain error</p> <p>nel linguaggio dell'informatica si dice che il programma dell'esempio precedente ha generato un'eccezione.</p> <p>La gestione strutturata delle eccezioni si ottiene racchiudendo gruppi di istruzioni che possono generare errori durante l'esecuzione e specificando le attività da svolgere quando l'eccezione viene rilevata e intercettata.</p> <p>La gestione strutturata è rappresentata dall'istruzione try...except..., secondo il seguente schema sintattico generale:</p> <pre>try: istruzioni1 except TipoErrore:</pre>

istruzioni2

la gestione delle eccezioni può contenere anche più *except* in corrispondenza di diversi tipi di errore.

Esempio

Il programma precedente può essere riscritto inserendo anche la gestione dell'eccezione:

```
import math
# input del radicando
n = float(input("Numero: "))
# calcola la radice quadrata
try:
    print('{:10.5f}'.format(math.sqrt(n)))
except ValueError:
    print("Il numero non è valido")
```

Quando si verifica la situazione di errore vengono eseguite le istruzioni scritte in modo indentato dopo *except*, inviando un messaggio di avvertimento all'utente.

I tipi di errore più comuni che si possono utilizzare con *except* sono i seguenti:

ValueError	Valore della variabile non corretto
NameError	Identificativo sconosciuto
ZeroDivisionError	Divisione per zero
FloatingPointError	Errore di calcolo tra numeri reali
OverflowError	Il calcolo ha superato i limiti di capacità di rappresentazione del dato numerico
TypeError	Operazione o funzione non valide per quel tipo di dato
UnboundLocalError	La variabile locale specificata non ha alcun valore assegnato
OSError	Errore di sistema

Esempio

```
a = 12
n = 0
try:
    print(a / n)
except ZeroDivisionError:
    print("Divisore nullo")
```

L'istruzione *try* completa, comprende anche la clausola **finally**, con la quale vengono specificate le istruzioni che vengono eseguite sempre prima di chiudere l'istruzione *try*, indipendentemente dal fatto che l'eccezione si sia verificata o meno.

```
try:
    istruzioni1
except TipoErrore:
    istruzioni2
finally:
    istruzioni3
```

	<p>Tipicamente finally serve per chiudere i file di dati aperti oppure per rilasciare risorse di elaborazione.</p> <p>Esempio a = 12 n = 0 try: print(a / n) except ZeroDivisionError: print("Divisore nullo") finally: print("Programma terminato")</p>						
La ripetizione <i>while</i> o ripetizione precondizionale	<p>La struttura di ripetizione realizzata con l'istruzione while ha la seguente sintassi: while condizione: blocco_istruzioni</p> <p>il carattere: nella riga di while indica l'inizio di un blocco di istruzioni indentate. Il segmento di codice precedente prevede che prima si controlla la condizione e, qualora sia verificata, si proceda all'esecuzione del blocco. La ripetizione continua fino a che la condizione è vera, appena diventa falsa si uscirà dal ciclo.</p>						
Coding	<p>La divisione tra interi Calcolare la divisione tra numeri interi usando le sottrazioni successive.</p> <table border="1"> <tr> <td> I: a, b O: quoziente </td><td> Legenda: a= dividendo b= divisore </td></tr> <tr> <td>Pseudo codifica</td><td>Programma Python (divisione.py)</td></tr> <tr> <td> Inizio (calcola) Quoziente=0 Chiedo, scrivo a, b Esegui mentre a>=b a=a-b quoziente= quoziente + 1 ripeti visualizza quoziente fine </td><td> # divisione.py : divisione tra interi con sottrazioni successive quoziente = 0 # input dei dati a = int(input("dividendo: ")) b = int(input("divisore: ")) # ripetizione delle sottrazioni e conteggio while a>= b: a = a - b quoziente = quoziente + 1 # output dei risultati print("quoziente = ", quoziente) print("resto =", a) </td></tr> </table>	I: a, b O: quoziente	Legenda: a= dividendo b= divisore	Pseudo codifica	Programma Python (divisione.py)	Inizio (calcola) Quoziente=0 Chiedo, scrivo a, b Esegui mentre a>=b a=a-b quoziente= quoziente + 1 ripeti visualizza quoziente fine	# divisione.py : divisione tra interi con sottrazioni successive quoziente = 0 # input dei dati a = int(input("dividendo: ")) b = int(input("divisore: ")) # ripetizione delle sottrazioni e conteggio while a>= b: a = a - b quoziente = quoziente + 1 # output dei risultati print("quoziente = ", quoziente) print("resto =", a)
I: a, b O: quoziente	Legenda: a= dividendo b= divisore						
Pseudo codifica	Programma Python (divisione.py)						
Inizio (calcola) Quoziente=0 Chiedo, scrivo a, b Esegui mentre a>=b a=a-b quoziente= quoziente + 1 ripeti visualizza quoziente fine	# divisione.py : divisione tra interi con sottrazioni successive quoziente = 0 # input dei dati a = int(input("dividendo: ")) b = int(input("divisore: ")) # ripetizione delle sottrazioni e conteggio while a>= b: a = a - b quoziente = quoziente + 1 # output dei risultati print("quoziente = ", quoziente) print("resto =", a)						

Osservazione:

Controllo sul divisore

Il programma precedente è eseguito correttamente purché il valore di b (divisore) sia diverso da zero.

In caso contrario la ripetizione while non termina mai perché la condizione si mantiene sempre vera.

Per evitare questa situazione occorre richiedere il valore di b finché viene inserito un valore diverso da zero, con una struttura while che controlla l'input del divisore:

```
while b == 0:  
    b = int(input("divisore: "))
```

Con questa precisazione, il programma precedente può essere riscritto in modo più completo con il seguente codice:

Programma Python (divisione2.py)

divisione2.py : divisione tra interi con sottrazioni successive

e controllo sul divisore

inizializzazione delle variabili

a = 0 **# dividendo**

b = 0 **# divisore**

quoziente = 0

input dei dati

a = int(input("dividendo: "))

while b == 0:

b = int(input("divisore: "))

ripetizione delle sottrazioni e conteggio

while a >= b:

a = a - b

quoziente = quoziente + 1

output dei risultati

print("quoziente =", quoziente)

print("resto =", a)

Gli operatori di assegnamento composti

Nelle operazioni di assegnamento si possono usare alcuni operatori che rappresentano in modo compatto le espressioni nel caso in cui si assegni un nuovo valore ad una variabile sulla base dell'attuale valore della stessa variabile.

Questi operatori si chiamano **operatori composti**.

operatori composti	Usato per:
+=	Incremento
-=	Decremento
*=	Moltiplicazione
/=	Divisione reale
//=	Divisione intera
%=	Resto della divisione tra interi
**=	Potenza

Esempio	
L'istruzione con operatori composti	Equivale a
totVendite += scontrino	totVendite = totVendite + scontrino
prezzo -= sconto	prezzo = prezzo – sconto
a -= b quoz += 1	a = a – b quoz = quoz + 1

n.b.
per incrementi e decrementi unitari si devono usare le istruzioni:
a += 1
a -= 1
poiché, a differenza di altri linguaggi di programmazione, Python non prevede gli operatori ++ e --.

Equivalenze tra i vari operatori di assegnamento

Operatore	Utilizzo	Esempio	Istruzione equivalente
+=	Incremento	x += y	x = x+y
-=	Decremento	x -= y	x = x-y
*=	Moltiplicazione	x *= y	x = x*y
/=	Divisione reale	x /= y	x = x/y
//=	Divisione intera	x //= y	x = x//y
%=	Resto della divisione tra interi	x %= y	x = x%y
=	Potenza	x **= y	x = xy

La ripetizione con controllo della fine dell'input

Nei problemi che riguardano l'input di un elenco di dati (numeri o stringhe), dei quali non si conosce a priori il numero, occorre segnalare la fine dell'input per terminare la ripetizione.

Se si volesse inserire da tastiera un elenco di numeri, segnando la fine dell'elenco con il **numero 0**, si può utilizzare il seguente frammento di programma:

```
numero = int(input("Inserisci il primo numero (0 = fine): "))
while numero != 0:
    .....
    .....
```

	<p>Numero = int(input("Inserire un altro numero (0 = fine): "))</p> <p>Il primo numero è inserito al di fuori della ripetizione while per poter effettuare il controllo della condizione numero !=0. Al termine del blocco indentato c'è un'altra istruzione di input prima di ripetere le istruzioni. Quando l'utente inserisce il valore 0, la condizione scritta vicino a while diventa falsa e la ripetizione termina.</p> <p>n.b.: questa organizzazione delle istruzioni offre all'utente la possibilità di non iniziare la ripetizione, nel caso sia stato avviato il programma in modo indesiderato: inserendo infatti al primo input il valore 0, la ripetizione non viene mai eseguita, perché la condizione risulta subito falsa.</p> <p>Se, invece, si vuole inserire un elenco di nomi (o più in generale di stringhe), usando il carattere * per segnalare la fine dell'elenco, si può scrivere:</p> <pre>nome = input("Inserisci il primo nome (* = fine): ") while nome != "*": nome = input("Inserire un altro nome (* = fine): ")</pre> <p>per controllare la fine dell'inserimento, il nome acquisito da tastiera viene confrontato con il carattere *.</p>
Coding	<p>Quanti hanno superato l'esame? Dato un elenco di studenti universitari, con l'indicazione per ciascuno di essi del nome e del voto conseguito in un esame, contare gli studenti che hanno superato l'esame.</p> <p>L'utente segnala la fine dell'input dei dati digitando un asterisco al posto del nome. Il nome del primo studente viene richiesto prima della ripetizione while e le operazioni dell'elenco sono eseguite mentre il nome è diverso da asterisco.</p> <p>Programma Python (elenco.py) # elenco.py: elenco di studenti con nome e voto VOTOMIN = 18 # voto minimo per superare l'esame conta = 0 # azzera il contatore # input del primo nome nome = input("Nome (* = fine): ") # ripetizione sull'elenco while nome != "*": voto = int(input("Voto: ")) if voto >= VOTOMIN: conta += 1 nome = input("Nome (* = fine): ") # output del risultato print("Hanno superato l'esame", conta, "studenti")</p>

Il problema viene scomposto in sottoprogrammi che svolge una specifica funzionalità per la risoluzione del problema; il programma viene scomposto in **parti funzionali indipendenti**.

Viene facilitato il lavoro di manutenzione del software, perché si può intervenire con modifiche o correzioni su un solo sottoprogramma, avendo allo stesso tempo cognizione di quello che fa l'intero programma.

Inoltre, alcuni sottoprogrammi essendo funzionalmente indipendenti possono essere riutilizzati, senza bisogno di grandi modifiche, all'interno di altri sottoprogrammi.

Nel linguaggio Python i sottoprogrammi sono rappresentati attraverso le **funzioni**.

Le funzioni usate finora (*int, float, oct, bin, hex, type, len*) sono esempi di **funzioni predefinite** o **funzioni built-in**, cioè unità software già disponibili al programmatore che, quindi, non richiedono la loro implementazione.

Una o più funzioni possono essere raggruppate in un file con estensione *.py*, che prende il nome di **modulo**, in modo da mettere a disposizione funzionalità utili in diversi programmi.

I moduli, o anche solo alcune delle funzioni in essi contenute, vengono inclusi in un programma mediante l'importazione con l'istruzione **import** (come la libreria *math* vista in precedenza).

Una **funzione** è un **procedimento** che riceve in input valori come **argomenti** e restituisce un **valore** come risultato. Si dice anche che la funzione **ritorna un valore**.

La sintassi generale di una funzione è:

```
def nome-funzione(parametri):  
    istruzioni  
    .....  
    return valore
```

la parola chiave **def** indica l'inizio della definizione di una funzione.

Dopo il nome della funzione, le parentesi tonde servono a contenere l'elenco dei **parametri formali**, separati da virgola, che saranno associati agli oggetti dal programma chiamante alla funzione.

Le istruzioni che formano il corpo della funzione sono scritte in modo indentato sotto la prima riga e rappresentano il codice che viene eseguito alla chiamata della funzione.

I valori restituiti

Nel caso in cui ci siano valori di ritorno, il blocco delle istruzioni della funzione contiene, come ultima istruzione, l'istruzione **return** seguita da un valore o dalla variabile contenente il risultato restituito dalla funzione.

L'istruzione *return* provoca il ritorno del controllo al programma principale o, in generale, alla funzione chiamante.

Esempio

La funzione che esegue la somma di due numeri interi, ricevuti come parametri, e restituisce il valore calcolato si scrive così:

```
def somma (a, b):  
    s = a + b  
    return s
```

in una funzione ci possono essere anche più *return*, in corrispondenza di uscite per diversi percorsi di elaborazione nella funzione.

Esempio

```
def limita_somma(a, b):  
    s = a + b  
    if s >= 10:  
        return s  
    else:  
        return 10
```

Se la funzione non restituisce alcun valore, si assume che il valore restituito sia **None** (nome predefinito del linguaggio). In questo caso l'istruzione *return*, alla fine della funzione, può essere omessa.

Esempio

```
def stampa_somma (a, b):  
    s = a + b  
    print(s)
```

La seguente funzione di stampa non restituisce alcun valore e non riceve alcun parametro. Le parentesi tonde vuote indicano che nessun parametro viene passato alla funzione:

```
def stampa():  
    print("stampa di prova")  
    print("fine della funzione")
```

n.b.:

Le funzioni che non restituiscono alcun valore sono più propriamente sottoprogrammi che servono a scomporre un programma complesso in modo strutturato. In altri linguaggi di programmazione sono indicate con il termine *procedure* o *subroutine*.

La chiamata della funzione

Per effettuare la **chiamata di una funzione** da un qualunque punto del programma, dopo averla definita, occorre specificare il nome della funzione seguito dall'elenco, tra parentesi tonde, degli argomenti da passare ai parametri della funzione:

nome funzione(argomenti)

Gli argomenti della funzione, separati dalla virgola, si chiamano anche **parametri attuali**.

Al momento della chiamata, l'interprete esegue il controllo di corrispondenza tra i parametri specificati nella definizione della funzione e i valori passati alla funzione.

	<p>Esempio</p> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <pre>def somma (a, b): s = a + b</pre> </div> <p style="color: red; margin-left: 400px;"># definizione della funzione somma</p> <pre>subtot1 = int (input("Parziale1: ")) subtot2 = int (input("Parziale2: ")) totale = somma(subtot1, subtot2) print(totale)</pre> <p style="color: red; margin-left: 350px;">#chiamata della funzione somma</p> <p>Il valore restituito da una funzione può essere assegnato ad una variabile, come nell'esempio precedente, oppure stampato con l'istruzione <i>print</i>, o ancora utilizzato in espressioni di calcolo.</p> <p>Esempio</p> <pre>print(somma(subtot1, subtot2)) perc = somma(subtot1, subtot2) / 100 * 10</pre> <p>se la funzione non ha parametri, la chiamata alla funzione non contiene alcun argomento tra parentesi tonde.</p> <p>Esempio</p> <pre>def stampa(): print("stampa di prova") print("fine della funzione") print(stampa())</pre> <p style="color: red;"># la funzione non ha valore di ritorno e quindi restituisce il valore predefinito None</p> <p>stampa di prova fine della funzione None</p>
Coding	<p>Equazione di secondo grado $ax^2+bx+c=0$</p> <p>Calcolare le soluzioni di un'equazione di secondo grado $ax^2+bx+c=0$.</p> <p>I: a,b,c O: soluzioni</p> <p>Analisi del problema: Dopo avere calcolato il discriminante Δ (delta) con la formula: $\Delta = b^2 - 4ac$</p> <p>Si possono verificare tre situazioni:</p> <ul style="list-style-type: none"> ✓ $\Delta < 0$ ✗ soluzioni reali ✓ $\Delta = 0$ $x_1 = x_2$ soluzioni reali e coincidenti ✓ $\Delta > 0$ $x_1 \neq x_2$ soluzioni reali e distinte <p style="text-align: center;">$x_{1,2} = (-b \pm \sqrt{\Delta}) / (2 * a)$</p> <p>Il programma deve prevedere tutti i casi quindi deve prevedere anche il caso in cui $a=0$, in questo caso l'equazione diventa di primo grado: $bx + c = 0$</p>

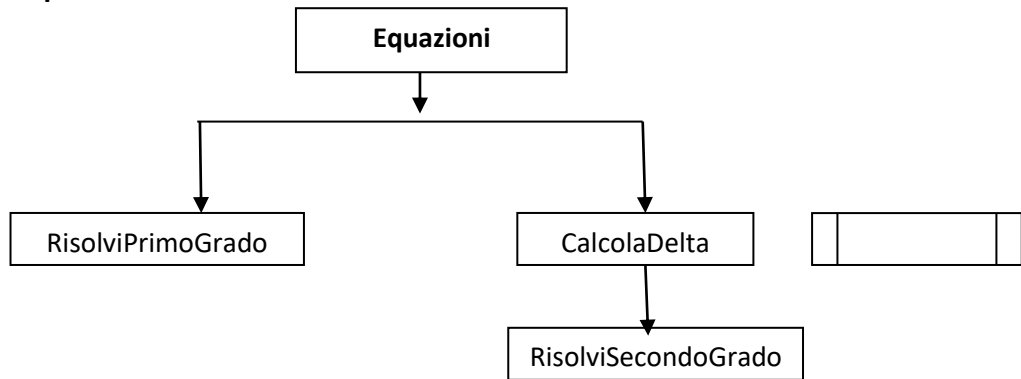
In questo caso può capitare che :

- $b=0$ and $c=0$ allora l'equazione è indeterminata;
- $b=0$ allora l'equazione è impossibile.

Da questa analisi si possono individuare tre **sottoprogrammi**:

- calcolo del Δ ;
- risoluzione dell'equazione di primo grado (se $a=0$);
- risoluzione dell'equazione di secondo grado quando $\Delta \geq 0$;

Top – down:



I sottoprogrammi rappresentano le macroistruzioni del procedimento risolutivo che vengono poi sviluppate in dettaglio.

Algoritmo Equazioni

Inizio (Equazioni)

Immetti a, b, c

Se $a \neq 0$

 RisolviSecondoGrado

Altrimenti

 RisolviPrimoGrado

Fine

Algoritmo RisolviPrimoGrado

Inizio (RisolviPrimoGrado)

Se $b = 0$

 Se $c = 0$

 Scrivi "equazione indeterminata"

 Altrimenti

 Scrivi "equazione impossibile"

Altrimenti

 Calcola $x = -c / b$

 Scrivi x

Fine

Algoritmo CalcoloDelta

Inizio (CalcoloDelta)

Calcola delta = $b^2 - 4*a*c$

Ritorna delta

Fine

Algoritmo RisolviSecondoGrado

Inizio (RisolviSecondoGrado)

Calcola delta

Se delta < 0

Visualizza “non esistono soluzioni reali”

Altrimenti

Calcola $x_1 = \frac{-b + \sqrt{\Delta}}{2 * a}$

Calcola $x_2 = \frac{-b - \sqrt{\Delta}}{2 * a}$

Visualizza x_1

Visualizza x_2

fine

I sottoprogrammi RisolviPrimoGrado e **RisolviSecondoGrado** sono richiamati **dal programma principale Equazioni** che è quindi il **programma chiamante**, tutti gli altri sottoprogrammi sono chiamati.

Il **sottoprogramma RisolviSecondoGrado**, a sua volta, richiama l'esecuzione del sottoprogramma CalcoloDelta.

Per **richiamare un sottoprogramma** basta scrivere il nome dello stesso all'interno del blocco istruzioni.

Programma Python (equazioni.py)

```

equazioni secondo grado.py - C:/Users/ANNA/Desktop/ex Python/equazioni secondo grado...
File Edit Format Run Options Window Help
# equazioni.py: soluzioni di un'equazione di secondo grado
import math # importazione delle librerie delle funzioni matematiche
# funzione per equazione di primo grado
def risolvi_primo_grado():
    if b == 0:
        if c == 0:
            print("Equazione indeterminata")
        else:
            print("Equazione impossibile")
    else:
        print("x =", -c / b)
# funzione per il calcolo del delta
def calcola_delta():
    d = math.pow(b, 2) - 4 * a * c
    return d
# funzione per equazione di secondo grado
def risolvi_secondo_grado():
    delta = calcola_delta()
    if delta < 0:
        print("Non esistono soluzioni reali")
    else:
        x1 = (-b + math.sqrt(delta)) / (2 * a)
        x2 = (-b - math.sqrt(delta)) / (2 * a)
        print("x1 =", x1)
        print("x2 =", x2)

# programma principale
a = float(input("Primo coefficiente: "))
b = float(input("Secondo coefficiente: "))
c = float(input("Terzo coefficiente: "))
if a != 0:
    risolvi_secondo_grado()
else:
    risolvi_primo_grado()

```

Le variabili globali e locali

Nel programma appena visto tutte le funzioni sono senza parametri. Le variabili *a*, *b*, *c*, inoltre, sono dichiarate al di fuori delle funzioni, perché sono utilizzate da più funzioni: per questo motivo si chiamano **variabili globali** in contrapposizione a quelle dichiarate all'interno delle funzioni, che prevedono il nome di **variabili locali**.

Nel programma precedente, sono variabili locali le variabili *d* nella funzione *calcola_delta* e le variabili *delta*, *x1* e *x2* nella funzione *risolvi_secondo_grado*.

La funzione *calcola_delta* esegue un'elaborazione e restituisce un valore al sottoprogramma chiamante tramite l'istruzione **return**.

Le funzioni con parametri

L'uso delle funzioni risponde all'esigenza fondamentale di costruire programmi ben organizzati e strutturati secondo la metodologia top-down.

Un'altra esigenza che viene risolta dalle funzioni è rappresentata dalla possibilità di utilizzare uno stesso gruppo di istruzioni in programmi diversi (come accade con le funzioni predefinite).

In questo caso conviene rendere parametrici i valori utilizzati dalle funzioni, in modo da ricevere come argomenti i valori passati dalla funzione chiamante.

La funzione main()

Quando i programmi saranno composti da più funzioni, useremo la convenzione di inserire nel programma una **funzione** particolare denominata **main()** come funzione principale, anche se non è strettamente richiesta dalle regole del linguaggio Python. Questa funzione non ha argomenti e non restituisce alcun valore.

L'avvio del programma sarà indicato con l'istruzione che chiama l'esecuzione della funzione *main()*, come ultima riga del programma: **main()**.

	<p>n.b.: E' buona norma scrivere le sottofunzioni prima della funzione main(). Poiché Python è un interprete che esamina il codice partendo dall'inizio e procedendo in avanti sarà bene scrivere in ordine di utilizzo le sottofunzioni presenti nel programma ovvero, bisognerà scrivere prima le sottofunzioni richiamate da altre funzioni e poi le funzioni chiamanti.</p>
Coding	<p>Equazione di secondo grado $ax^2+bx+c=0$ Calcolare le soluzioni di un'equazione di secondo grado $ax^2+bx+c=0$ introducendo i parametri nelle funzioni.</p> <p>In questa versione del programma le funzioni hanno nella definizione, tra le parentesi tonde, l'elenco dei parametri che sono associati agli argomenti delle funzioni chiamate.</p> <p>Programma Python (equazioni2.py)</p>

```

*equazioni2 secondo grado.py - C:/Users/ANNA/Desktop/ex Python/equazioni2 secondo gr...
File Edit Format Run Options Window Help
# equazioni2.py: soluzioni di un'equazione di secondo grado
# con uso dei parametri

import math      # importazione delle libreria delle funzioni matematiche

# funzione per equazione di primo grado
def risolvi_primo_grado(c2, c3):
    if c2 == 0:
        if c3 == 0:
            print("Equazione indeterminata")
        else:
            print("Equazione impossibile")
    else:
        print("x =", -c3 / c2)

# funzione per il calcolo del delta
def calcola_delta(c1, c2, c3):
    d = math.pow(c2, 2) - 4 * c1 * c3
    return d

# funzione per equazione di secondo grado
def risolvi_secondo_grado(c1, c2, c3):
    delta = calcola_delta(c1, c2, c3)
    if delta < 0:
        print("Non esistono soluzioni reali")
    else:
        x1 = (- c2 + math.sqrt(delta)) / (2 * c1)
        x2 = (- c2 - math.sqrt(delta)) / (2 * c1)
        print("x1 =", x1)
        print("x2 =", x2)

# programma principale
a = float(input("Primo coefficiente: "))
b = float(input("Secondo coefficiente: "))
c = float(input("Terzo coefficiente: "))
if a != 0:
    risolvi_secondo_grado(a, b, c)
else:
    risolvi_primo_grado(b, c)

# istruzione di avvio del programma

```

Ln: 25 Col: 0

n.b.:

Il vantaggio di questa modalità d'uso delle variabili e dei parametri consiste nel fatto che le funzioni così strutturate diventano unità software che possono essere utilizzate trasportando senza modifiche il loro codice in altri programmi che richiedono le stesse funzionalità (facendo copia e incolla oppure usando i moduli, che vedremo in seguito).

Parametri formali e attuali

Le variabili indicate nella definizione della funzione, tra parentesi tonde dopo il nome della funzione, si chiamano **parametri formali**: nel Coding precedente, sono **parametri formali c1, c2, c3** nelle funzioni *calcola_delta* e *risolvi_secondo_grado* etc...

Le variabili che sono gli argomenti nella chiamata della funzione e forniscono i valori ai parametri si chiamano **parametri attuali** (nel Coding precedente le variabili a, b, c).

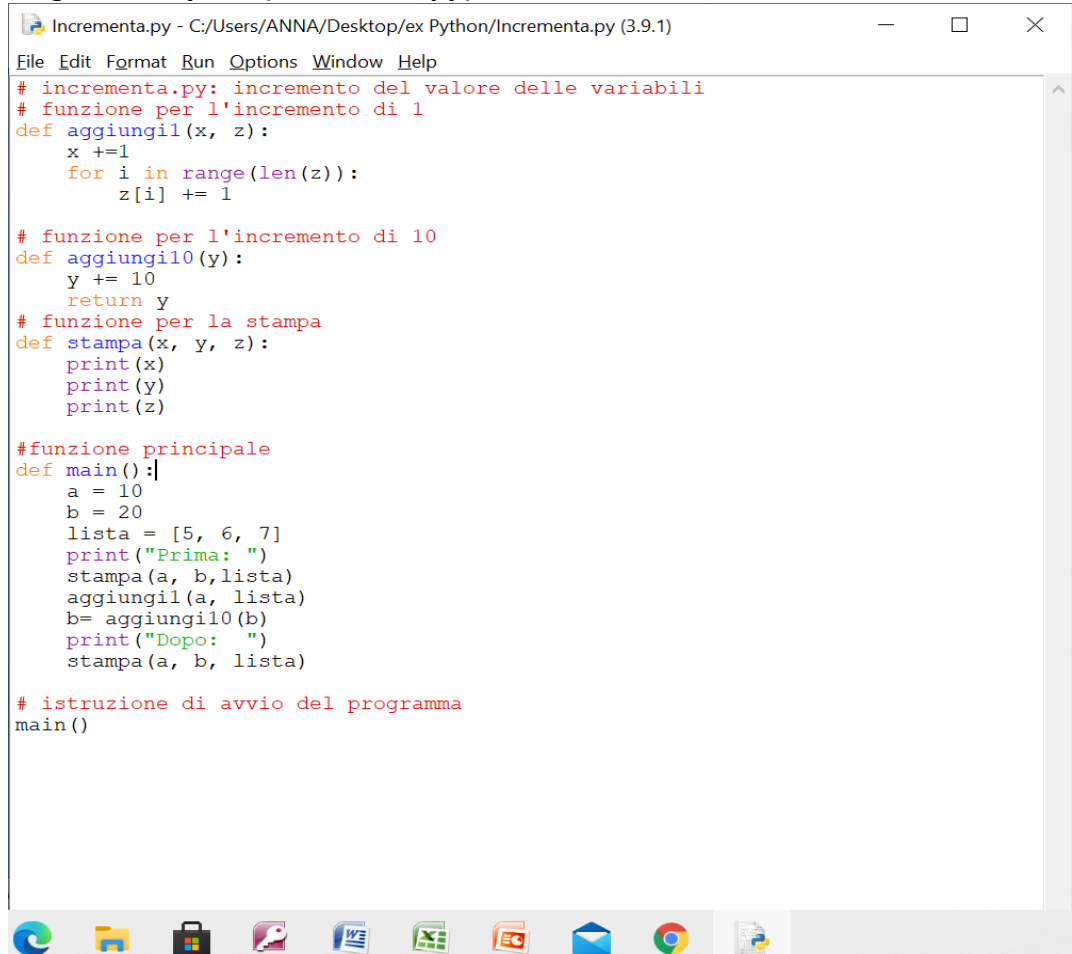
Nell'uso delle funzioni occorre rispettare **tre regole di corrispondenza** tra parametri attuali e formali:

- corrispondenza di **tipo**: il tipo di dato del parametro formale deve essere lo stesso tipo del parametro attuale;
- corrispondenza di **numero**: il numero degli argomenti forniti dalla funzione chiamante deve essere uguale al numero dei parametri previsti nella definizione della funzione ;
- corrispondenza in **ordine posizionale**: il primo argomento nella chiamata della funzione chiamante deve essere associato al primo parametro formale, il secondo al secondo parametro e così via.

	<p>Esempio Se nel Coding precedente si scrive la chiamata di una funzione senza argomenti: <pre>if a != 0: risolvi_secondo_grado()</pre> si ottiene il seguente messaggio di errore: Traceback (most recent call last): File "C:/Users/ANNA/Desktop/equazione con errore.py", line 37, in <module> risolvi_secondo_grado() TypeError: risolvi_secondo_grado() missing 3 required positional arguments: 'c1', 'c2', and 'c3'</p> <p>L'interprete Python, in fase di esecuzione, segnala che la definizione della funzione richiede tre argomenti da associare ai parametri.</p> <p>Volendo passare i valori in un ordine diverso da quello usato nella definizione, è sufficiente effettuare la chiamata esplicitando il nome del parametro formale seguito dal simbolo di assegnazione (=) e dal valore del parametro attuale.</p> <p>Esempio Se nel Coding precedente si scrive la chiamata di una funzione specificando i parametri formali: <pre>if a != 0: risolvi_secondo_grado(c2=b, c3=c, c1=a)</pre> si ottiene lo stesso comportamento della chiamata ordinata: <pre>if a != 0: risolvi_secondo_grado(a, b, c)</pre></p>
Il passaggio degli argomenti ai parametri	<p>L'operazione con la quale il programma chiamante invia gli oggetti alla funzione, assegnandoli ai nomi dei parametri, si chiama passaggio degli argomenti ai parametri. Si dice anche che gli argomenti sono passati ai parametri per assegnamento.</p> <p>Nella chiamata alla funzione, viene assegnata al parametro la referenza all'oggetto (object reference) rappresentato dall'argomento (<i>call by object reference</i>), cioè il riferimento all'indirizzo della memoria centrale che contiene l'oggetto.</p> <p>Se l'oggetto passato è un intero o una stringa (oggetto immutabile), il suo valore non cambia durante l'esecuzione della funzione. Gli oggetti passati diventano identificatori locali della funzione e non c'è alcun collegamento con le variabili della funzione chiamante: di fatto viene fatta una copia dell'oggetto.</p> <p>Se, invece, l'oggetto passato è una lista (oggetto mutabile), gli eventuali cambiamenti di valore ai parametri, durante l'esecuzione della funzione, influenzano i valori delle variabili corrispondenti nella funzione chiamante.</p>
Coding	<p>L'incremento di numeri Utilizzare le funzioni per incrementare il valore di alcune variabili confrontando i diversi risultati che si ottengono nel passaggio degli argomenti ai parametri della funzione.</p> <p>Il programma utilizza due variabili <i>a</i>, <i>b</i> di tipo intero ed una lista di numeri interi. La prima funzione aggiungi1() riceve <i>a</i> e <i>lista</i>, che vengono assegnate ai parametri <i>x</i> e</p>

z, ed effettua l'incremento di 1 sia sulla variabile sia sugli elementi della lista. La **seconda funzione** *aggiungi10()* riceve *b*, che viene assegnata al parametro *y*, incrementa *y* di 10 e restituisce il valore incrementato. Una **terza funzione**, *stampa()*, serve a visualizzare il contenuto degli oggetti prima e dopo la chiamata delle funzioni, per evidenziare il diverso effetto del passaggio degli argomenti ai parametri della funzione.

Programma Python(*incrementa.py*)



```
# incrementa.py: incremento del valore delle variabili
# funzione per l'incremento di 1
def aggiungi1(x, z):
    x += 1
    for i in range(len(z)):
        z[i] += 1

# funzione per l'incremento di 10
def aggiungi10(y):
    y += 10
    return y
# funzione per la stampa
def stampa(x, y, z):
    print(x)
    print(y)
    print(z)

#funzione principale
def main():
    a = 10
    b = 20
    lista = [5, 6, 7]
    print("Prima: ")
    stampa(a, b, lista)
    aggiungi1(a, lista)
    b = aggiungi10(b)
    print("Dopo: ")
    stampa(a, b, lista)

# istruzione di avvio del programma
main()
```

Il parametro *a* è stato copiato nel parametro *x*, che è un identificatore locale della funzione *aggiungi1()*: l'incremento all'interno della funzione non influisce sul valore di *a*, essendo un oggetto immutabile.

I valori degli elementi di *lista* vengono incrementati nella funzione *aggiungi1()* e l'incremento si riflette sulla lista della funzione chiamante, essendo la lista un oggetto mutabile.

Il valore di *b* viene passato al parametro *y*, che viene incrementato nella funzione *aggiungi10()*: alla variabile *b* viene assegnato il valore restituito dalla funzione e risulta quindi incrementato di 10.

n.b.:

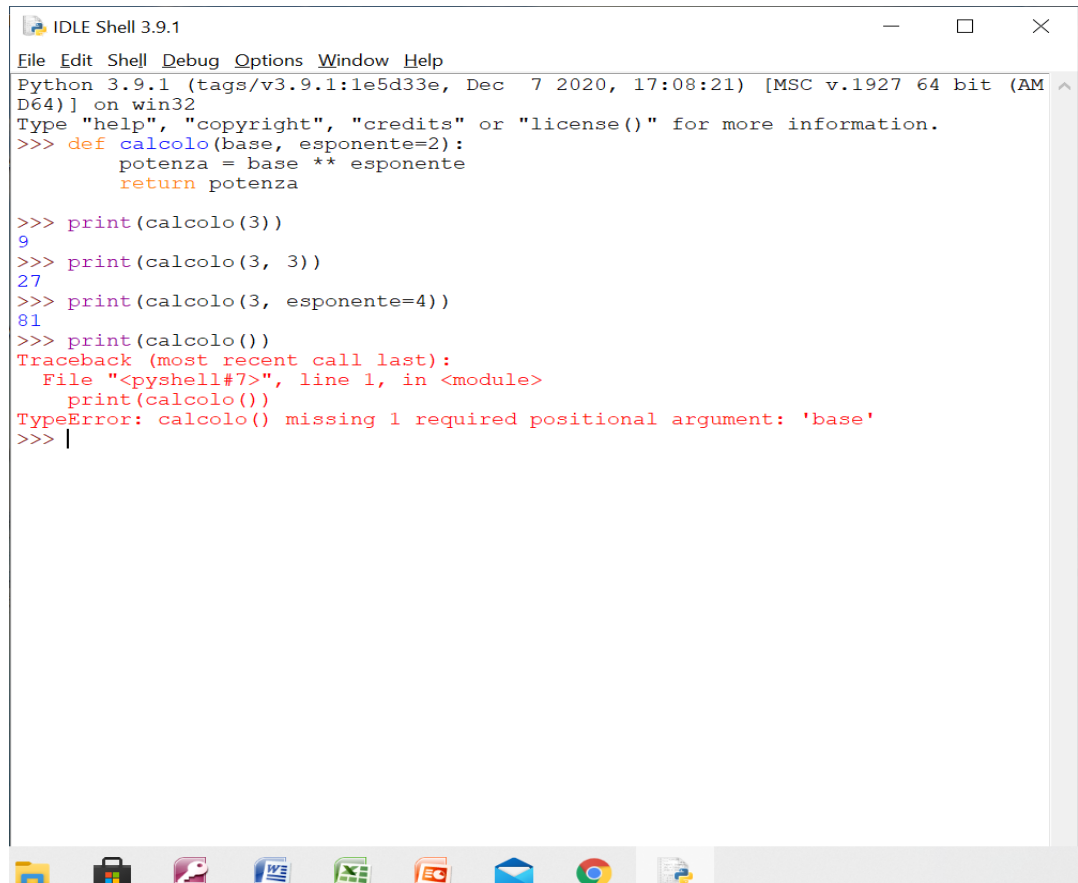
L'uso della funzione *stampa()* evidenzia l'importanza dell'organizzazione di un programma in modo strutturato con le funzioni ed i parametri: una stessa funzione può essere richiamata in più punti del programma con valori diversi assegnati ai parametri.

I parametri di default

La definizione di una funzione può contenere alcuni parametri obbligatori ed alcuni opzionali. La chiamata della funzione richiede la specificazione di un numero di argomenti pari al numero dei parametri obbligatori; se non vengono forniti gli argomenti opzionali, la funzione assume i valori di *default* previsti per i parametri nella definizione.

I **parametri di default** sono scritti nella forma *identificativo= valore*.

Esempio



```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> def calcolo(base, esponente=2):
    potenza = base ** esponente
    return potenza

>>> print(calcolo(3))
9
>>> print(calcolo(3, 3))
27
>>> print(calcolo(3, esponente=4))
81
>>> print(calcolo())
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    print(calcolo())
TypeError: calcolo() missing 1 required positional argument: 'base'
>>> |
```

- Nella prima elaborazione viene assunto l'esponente di default (2).
- Nella seconda gli argomenti vengono assegnati a entrambi i parametri usando la notazione posizionale.
- Nella terza elaborazione viene specificato il nome del parametro ed il valore da assegnare (*keyword argument*)

Sappiamo che la funzione predefinita **range()** indica un intervallo di valori. Essa ha tre argomenti *start*, *stop*, *step*, di cui il secondo è obbligatorio e gli altri due sono opzionali.

Esempio

Si possono scrivere le seguenti istruzioni:

for i in range(6): intervallo di 6 valori a partire da 0

for i in range(1, 6): intervallo da 1 a 6 escluso

	<p>for i in range(1, 15, 3): intervallo da 1 a 15 escluso, con passo 3</p>
<p>La visibilità degli oggetti e i namespace</p>	<p>L'idea di organizzare i programmi in sottoprogrammi funzionalmente indipendenti porta con sé la necessità di definire, all'interno della funzione, tutte le risorse necessarie al suo funzionamento.</p> <p>Il termine risorsa sta ad indicare un qualsiasi oggetto utilizzabile dal programma o dalle singole funzioni: costanti, variabili, funzioni definite dal programmatore, funzioni di libreria del linguaggio Python.</p> <p>In un programma si possono distinguere:</p> <ul style="list-style-type: none"> • Oggetti locali, che vengono dichiarati all'interno della funzione che le utilizza e che non sono visibili alle altre funzioni; • Oggetti globali, che vengono dichiarati al di fuori delle funzioni; • Oggetti built-in, che sono predefiniti nel linguaggio. <p>Esempio</p> <pre> nome = valore oggetti globali def funzione1(parametri): x = valore1 oggetti locali della funzione print(math.sqrt(x)) oggetto built-in del linguaggio def main(): y = valore2 main() </pre> <p>Sulla base del ragionamento fatto in precedenza, un programma ben strutturato (cioè scritto in modo chiaro, efficace ed efficiente) avrà poche variabili globali e molte variabili locali per consentire una facile manutenzione del software nel tempo e maggiori possibilità di riutilizzare moduli software in altri programmi.</p> <p>Un esempio tipico di variabile locale è costituito da contatori, o in generale da variabili di lavoro, la cui funzionalità si esaurisce all'interno della funzione e che è quindi inutile dichiarare come variabili globali.</p> <p>Per le variabili locali vengono riservate aree di MC solo al momento dell'esecuzione della funzione. Lo spazio viene creato all'inizio e rilasciato al termine della funzione, liberando quindi spazio nella memoria utente.</p> <p>Questo consente di utilizzare lo stesso nome di variabile locale in due funzioni diverse.</p>
<p>Il namespace</p>	<p>E' una tabella che fa corrispondere (<i>mapping</i>) gli identificativi agli oggetti. Quindi un programma dispone del <i>namespace</i> degli oggetti locali, del <i>namespace</i> degli oggetti globali e del <i>namespace</i> degli identificativi per gli oggetti <i>built-in</i>.</p> <p>L'ordine di ricerca è:</p> <ol style="list-style-type: none"> 1. prima nella tabella del <i>namespace</i> locale, 2. poi nel <i>namespace</i> globale

	3. infine nel <i>namespace</i> degli identificativi <i>built-in</i> .
La visibilità	<p>La visibilità (<i>scope</i>) di un oggetto, ossia la disponibilità di un oggetto all'interno del programma, è specificata nel <i>namespace</i>, in cui si trova la corrispondenza tra il nome dell'oggetto e l'oggetto stesso.</p> <p>Le regole di visibilità (<i>scope rules</i>) sono i criteri con i quali si stabilisce se un programma o una funzione <i>vede</i> un oggetto, nel senso che può utilizzare l'oggetto. Le regole di visibilità possono essere riassunte nel seguente modo. Ogni ambito vede:</p> <ul style="list-style-type: none"> • le proprie risorse locali • le risorse definite all'esterno della definizione delle funzioni. <p>In sostanza una funzione può utilizzare tutti gli oggetti dichiarati localmente, gli oggetti globali dichiarati al di fuori della definizione della funzione (che hanno nomi diversi dagli oggetti locali) e gli oggetti predefiniti del linguaggio.</p> <p>Di contro, una funzione non può usare gli oggetti definiti localmente in altre funzioni.</p> <p>Esempio</p> <p>Nel seguente programma le funzioni vedono i propri oggetti locali, la variabile globale e la funzione <i>built-in</i> per il calcolo della radice quadrata (<i>math.sqrt</i>), ma la funzione <i>main()</i> non vede la variabile <i>b</i> che è un oggetto locale alla funzione <i>stampa()</i>.</p> <pre>import math a = 7 def stampa(): b = 5 print ("nella funzione stampa: ") print (b) print (a) print(math.sqrt(144)) def main(): c = 10 stampa() print ("nella funzione main: ") print (c) print (a) print(b) main()</pre>

Lo shadowing

Cosa succede se la variabile locale e la variabile globale hanno lo stesso nome?
 Il codice seguente usa lo stesso nome di variabile per una variabile globale e per una variabile locale di tipo diverso: questa situazione viene indicata in inglese con il termine **shadowing** (letteralmente “mettere in ombra”).

Esempio

x = 10.32

```
def f():
    x = 10
    print(x) # visualizza 10: la variabile locale x di tipo int mette in ombra la variabile globale x di tipo float
```

```
def main():
    print(x) # visualizza 10.32
    f()
    print(x) # visualizza 10.32: l'esecuzione di f() non modifica il valore della variabile globale x
```

main ()

n.b. :

Una funzione non è in grado di modificare il valore di una variabile globale. Infatti, l'assegnazione per modificare il valore di x in 10 ha semplicemente creato una nuova variabile locale. Le variabili globali sono quindi disponibili solamente in lettura all'interno delle funzioni.

La parola chiave **global**

Le limitazioni viste in precedenza possono essere superate attraverso una dichiarazione con la parola chiave **global**

	<p>global <i>nome variabile</i></p> <p>Con questa istruzione si può:</p> <ul style="list-style-type: none"> • consentire l'assegnamento di un valore alla variabile globale dall'interno di una funzione, eliminando l'effetto dello <i>shadowing</i>; • rendere disponibile all'esterno una variabile dichiarata all'interno di una funzione. <p>Esempio</p> <p>Nella funzione <i>f1()</i> si effettua l'assegnamento di un valore alla variabile globale <i>x</i>, mentre normalmente una funzione può solo usare una variabile globale, ma non può cambiarne il valore.</p> <p>Nella funzione <i>f2()</i> viene dichiarata una variabile locale che viene resa disponibile a tutto il programma.</p> <pre> x = 20.1 def f1(): global x x = 10 print (x) # visualizza 10 def f2(): global y y = 3 def main(): print (x) # visualizza 20.1 f1() print (x) # visualizza 10: l'esecuzione di f1() modifica il valore di x f2() print (y) # visualizza 3: la funzione main() vede la variabile y dichiarata all'interno di f2() main () </pre>
<p>La funzione lambda</p>	<p>Nel linguaggio Python si possono definire le funzioni anonime attraverso la parola chiave lambda. Per questo motivo si chiamano funzioni lambda.</p> <p>Queste funzioni sono anonime perché, a differenza di quelle viste finora e definite con la parola chiave <i>def</i>, non hanno un nome: hanno solo gli argomenti ed un'unica espressione di calcolo.</p> <p>La struttura generale della funzione <i>lambda</i> è la seguente:</p> <p>lambda argomenti: <i>espressione</i></p> <p>la definizione della funzione è scritta tutta su una riga (funzione <i>inline</i>)</p>

n.b.:

Le caratteristiche descritte evidenziano che queste funzioni risultano utili quando si vuole inserire in modo rapido nel programma un'elaborazione di tipo funzionale, cioè che riceve degli argomenti e che restituisce un valore, e non c'è la necessità di definire una funzione richiamabile con un nome specifico.

Esempio

La seguente funzione *lambda* calcola il doppio di un numero, con la scrittura del codice in modo compatto:

```
>>> doppio = lambda x: x * 2
>>> print (doppio(3))
6
```

Essa produce lo stesso risultato della seguente definizione di funzione:

```
>>> def doppio(x):
    return x * 2
>>> print(doppio(3))
6
```

L'istruzione *print* può essere scritta in modo ancora più compatto con la seguente riga di codice:

```
>>> print ((lambda x: x * 2 )(3))
6
```

Se gli argomenti sono più di uno, sono scritti uno di seguito all'altro, dopo la parola *lambda*, separati da virgola.

Esempio

Calcolo dell'IVA: la funzione *lambda* riceve come argomenti l'imponibile e l'aliquota IVA e restituisce l'importo dell'imposta:

```
>>> imposta = lambda imponibile, aliquota: imponibile/100 * aliquota
>>> print(imposta(200, 22))
44.0
```

Divisibilità di un numero: la funzione *lambda* controlla se un numero *x* è divisibile per un numero *y* (l'espressione di calcolo contiene l'operatore % per il calcolo del resto):

```
>>> divisibile = lambda x,y: x%y ==0
>>> print(divisibile(60,12))
True
```

Le funzioni predefinite

Il linguaggio Python possiede molte **funzioni predefinite** (*built-in*), che il programmatore può usare senza la loro definizione.

La tabella seguente spiega il significato di alcune funzioni di uso comune:

Funzioni di uso generale	
input	Acquisisce un dato da standard input come stringa di caratteri
print	Visualizza dati e messaggi su standard output
help	Attiva l'help predefinito del linguaggio
len	Restituisce la lunghezza di un oggetto
map	Elabora gli elementi di una lista e restituisce un insieme di dati
filter	Elabora una lista ed estrae un insieme di dati
list	Converte un insieme di dati in una lista
type	Restituisce la <i>class</i> di un oggetto (tipo di dato)
range	Restituisce un intervallo di valori
open	Apre un file
exec	Esegue il codice Python

n.b.:

per eseguire uno script *prova.py* dalla *Shell* di IDLE si può usare la funzione **exec** con un comando come il seguente:

```
>>> exec(open('prova.py').read())
```

L'interprete esegue l'accesso in lettura al file *prova.py* con la funzione predefinita *open*: essa restituisce un oggetto di tipo **file**, al quale viene applicato il metodo **read**. Il nome dello script è una stringa specificata come argomento della funzione *open*.

Funzioni di conversione	
bin(x)	Converte un numero intero <i>x</i> in <i>binario</i>
chr(x)	Restituisce il codice Unicode corrispondente al valore di <i>x</i>
complex(x,y)	Restituisce il numero complesso $x + yj$
float(x)	Converte un numero o una stringa in <i>float</i>
hex(x)	Converte un numero intero <i>x</i> in <i>esadecimale</i>
int(x)	Converte un numero o una stringa in <i>int</i>
oct(x)	Converte un numero intero <i>x</i> in <i>ottale</i>
ord(c)	Restituisce il codice Unicode di un carattere <i>c</i> (funzione inversa di <i>chr</i>)
repr(x)	Restituisce una stringa con la rappresentazione stampabile di un oggetto

Funzioni di calcolo	
abs(x)	Calcola il valore assoluto di un numero
max(x1,x2,x3,...)	Restituisce il massimo in un insieme di valori
min(x1,x2,x3,...)	Restituisce il minimo in un insieme di valori
pow(x,y)	Calcola la potenza intera di x^y (con <i>x</i> e <i>y</i> interi)
round(x, n)	Arrotonda un numero <i>x</i> con <i>n</i> cifre decimali

Le funzioni matematiche

Le **funzioni matematiche** sono disponibili nel programma dopo l'importazione della libreria **math** (modulo).

L'istruzione per l'importazione è:

```
import math
```

Alcune funzioni della libreria *math* sono:

Funzioni matematiche	
ceil(x)	Il più piccolo intero maggiore o uguale a x
exp(x)	Esponenziale di x (e^x)
fabs(x)	Valore assoluto di x come numero <i>float</i>
floor(x)	Il più grande numero intero minore o uguale a x
log(x)	Logaritmo in base e
log10(x)	Logaritmo in base 10
pow(x,y)	Potenza di x^y come numero float
sqrt(x)	Radice quadrata
trunc(x)	Valore troncato alla parte intera
sin, cos, tan, acos, asin, atan	Funzioni trigonometriche (l'argomento deve essere espresso in radianti)
degrees(x)	Conversione da radianti a gradi
radians(x)	Conversione da gradi a radianti

La libreria *math* fornisce anche due **costanti**:

Costanti	
math.pi	Costante $\pi = 3.141592$
math.e	Costante $e = 2.718281$

n.b.:

Nel calcolo del valore assoluto di un numero, la funzione **math.fabs(x)** restituisce sempre un numero *float*, anche se l'argomento è intero, mentre la funzione **abs()** restituisce un *float* o un *int* a seconda del tipo di argomento.

Una considerazione analoga vale per la differenza tra la funzione **math.pow()** e la funzione **pow()**. Quindi per ottenere la potenza intera esatta, con base ed esponente interi, occorre usare la funzione **pow()** che è equivalente all'operatore ******.

```

IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help()", "copyright()", "credits()" or "license()" for more information.
>>> abs(-5)
5
>>> 3 ** 4
81
>>> pow(3, 4)
81
>>> pow(1.2, 3)
1.7279999999999998
>>> import math
>>> math.fabs(-5)
5.0
>>> math.pow(3, 4)
81.0
>>> |
  
```

Coding

Il punto e la retta

Calcolare la distanza di un punto P(x, y) da una retta $ax+by+c=0$

Dati di input: i coefficienti a, b, c dell'equazione della retta nella forma $ax + by + c = 0$; le coordinate (x0, y0) del punto P.

Dati di output: la misura della distanza del punto P dalla retta.

La distanza si calcola con la seguente formula: $d = |ax_0 + by_0 + c| / \sqrt{a^2 + b^2}$

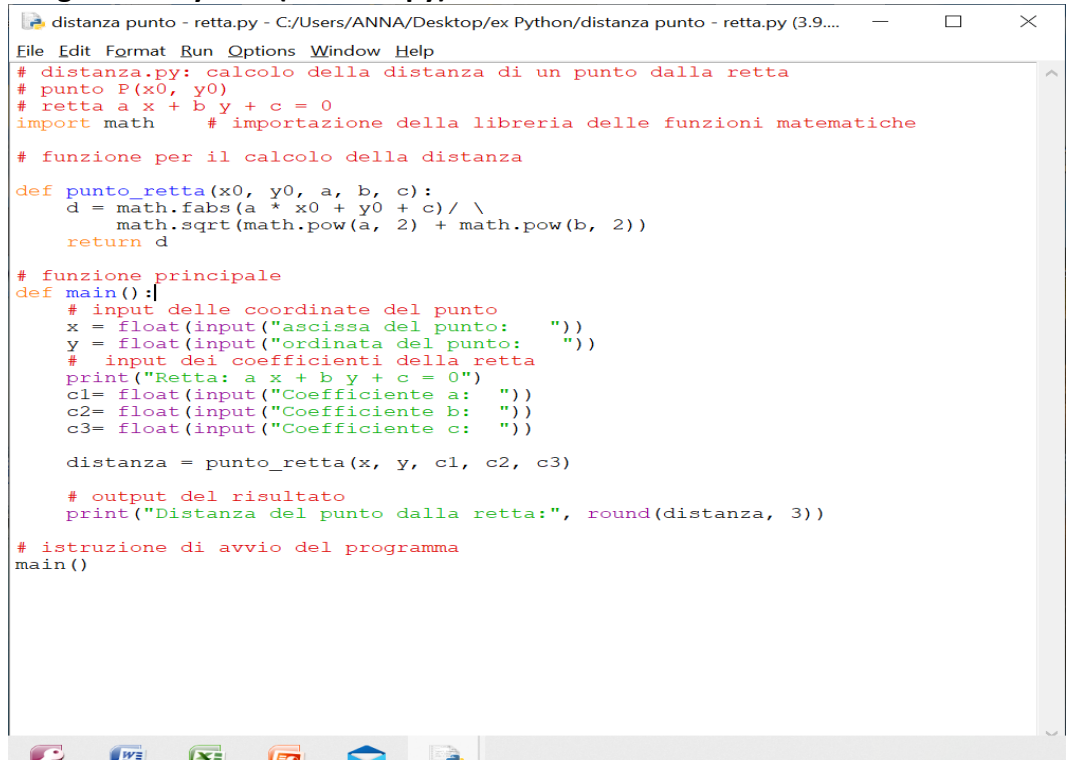
La funzione `punto_retta` per il calcolo della distanza richiede cinque parametri che rappresentano le coordinate del punto e i coefficienti dell'equazione della retta.

La funzione utilizza le funzioni predefinite della libreria **math** del linguaggio Python:

- **math.fabs()** per il calcolo del valore assoluto di un numero o di una espressione;
- **math.sqrt()** per il calcolo della radice quadrata;
- **math.pow()** per il calcolo della potenza.

Nella funzione principale, la funzione **round()** nella funzione `print()` consente di visualizzare il risultato con tre cifre decimali.

Programma Python (distanza.py)



```
distanza punto - retta.py - C:/Users/ANNA/Desktop/ex Python/distanza punto - retta.py (3.9...
File Edit Format Run Options Window Help
# distanza.py: calcolo della distanza di un punto dalla retta
# punto P(x0, y0)
# retta a x + b y + c = 0
import math # importazione della libreria delle funzioni matematiche

# funzione per il calcolo della distanza

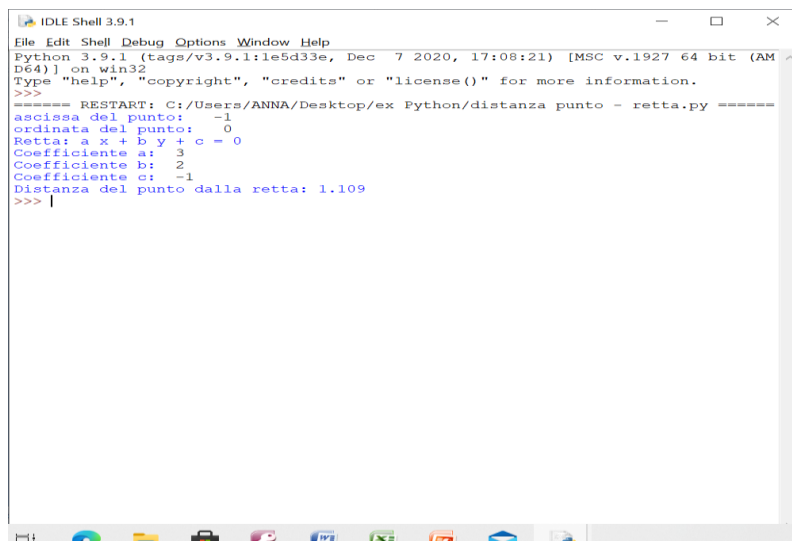
def punto_retta(x0, y0, a, b, c):
    d = math.fabs(a * x0 + b * y0 + c) / \
        math.sqrt(math.pow(a, 2) + math.pow(b, 2))
    return d

# funzione principale
def main():
    # input delle coordinate del punto
    x = float(input("ascissa del punto: "))
    y = float(input("ordinata del punto: "))
    # input dei coefficienti della retta
    print("Retta: a x + b y + c = 0")
    c1 = float(input("Coefficiente a: "))
    c2 = float(input("Coefficiente b: "))
    c3 = float(input("Coefficiente c: "))

    distanza = punto_retta(x, y, c1, c2, c3)

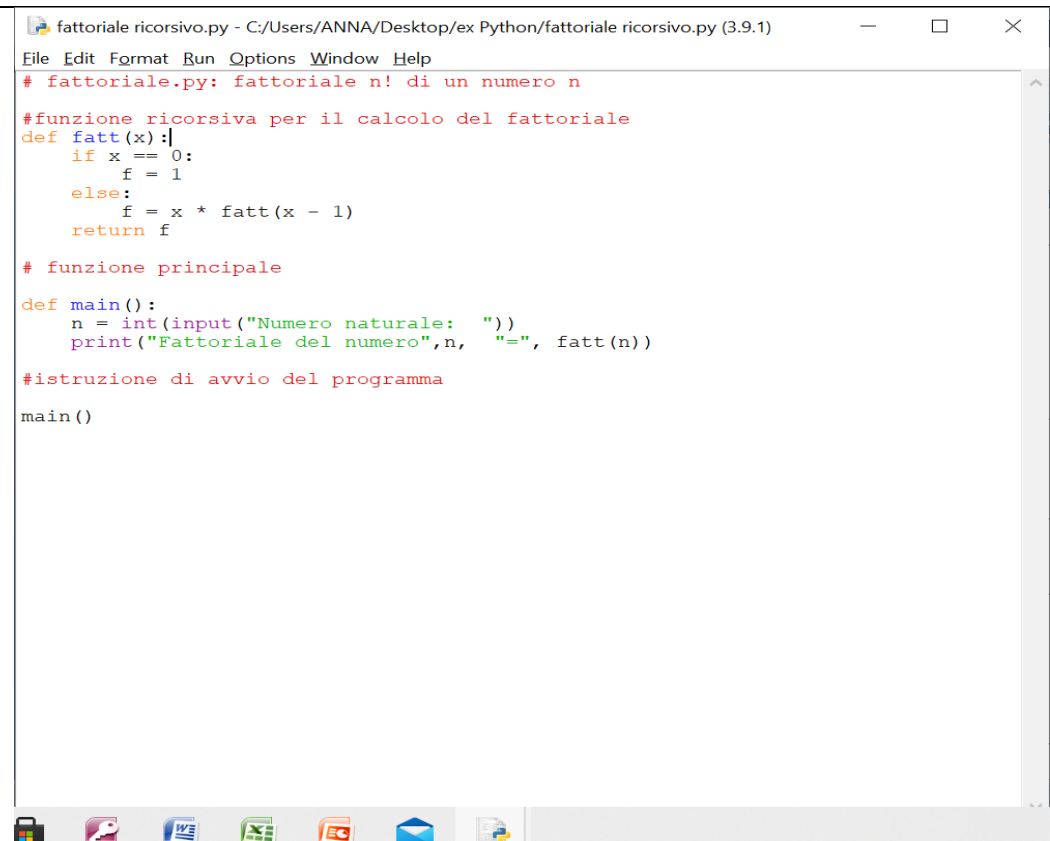
    # output del risultato
    print("Distanza del punto dalla retta:", round(distanza, 3))

# istruzione di avvio del programma
main()
```



```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/ANNA/Desktop/ex Python/distanza punto - retta.py =====
ascissa del punto: -1
ordinata del punto: 0
Retta: a x + b y + c = 0
Coefficiente a: 3
Coefficiente b: 2
Coefficiente c: -1
Distanza del punto dalla retta: 1.109
>>> |
```

<p>La ricorsione</p>	<p>Con il termine ricorsione (o <i>ricorsività</i>) si indica un algoritmo di calcolo che è in grado di chiamare l'esecuzione del calcolo stesso, cioè la chiamata dell'algoritmo è un'istruzione dello stesso algoritmo.</p> <p>In matematica possono essere definiti procedimenti di calcolo ricorsivo quali il calcolo del fattoriale di un numero o il calcolo della potenza di un numero, come mostrato nei <i>Coding</i> successivi.</p> <p>Nei linguaggi di programmazione la ricorsione si riferisce alle funzioni che hanno la possibilità di chiamare se stesse, cioè la chiamata della funzione è contenuta all'interno della funzione stessa. In sostanza la funzione diventa essa stessa una risorsa per l'attività di elaborazione, insieme alle variabili, ai parametri e ad altre funzioni.</p> <p>La ricorsione è possibile perché, ad ogni chiamata della funzione, viene generata in MC una copia della funzione, come se fosse una funzione diversa. Il codice relativo ad una funzione viene caricato in MC al momento della chiamata e rilasciato appena termina l'esecuzione della funzione: questa attività del SO, nella gestione della MC, si chiama allocazione dinamica del codice.</p> <p>n.b.: in precedenza, è stato spiegato che, alla chiamata di una funzione, viene creato un <i>namespace</i> locale per la funzione (vedi pag. 52). Nel caso della chiamata ricorsiva, ogni chiamata genera un proprio <i>namespace</i> locale.</p>
<p>Coding</p>	<p>Il fattoriale di un numero n (n!) Calcolare il fattoriale con un procedimento ricorsivo.</p> <p>n! si legge fattoriale di n e sta ad indicare la moltiplicazione di tutti gli interi minori o uguali a n:</p> $n! = n * (n-1) * (n-2) * \dots * 3 * 2 * 1$ <p>per definizione $0! = 1$ $1! = 1$</p> <p>È possibile per il calcolo del fattoriale dare anche una definizione ricorsiva:</p> <ul style="list-style-type: none"> • per n=0 e per n=1, $n!=1$ • per n>0, $n! = n * (n-1)$ <p>questa definizione viene usata per realizzare una funzione ricorsiva <i>fatt(x)</i> che calcola il fattoriale di x.</p> <p>Nella funzione <i>fatt()</i> è contenuta la chiamata della funzione stessa, passando al parametro il valore di x-1.</p> <p>Programma Python (fattoriale.py)</p>



```
# fattoriale.py: fattoriale n! di un numero n

#funzione ricorsiva per il calcolo del fattoriale
def fatt(x):
    if x == 0:
        f = 1
    else:
        f = x * fatt(x - 1)
    return f

# funzione principale
def main():
    n = int(input("Numero naturale: "))
    print("Fattoriale del numero", n, "=", fatt(n))

#istruzione di avvio del programma
main()
```

n.b.:

La libreria *math* delle funzioni matematiche comprende anche la funzione **math.factorial()** per il calcolo del fattoriale di un numero.

La funzione restituisce il fattoriale del numero x fornito come argomento, oppure un messaggio di errore *ValueError* se x non è intero o se è negativo.

La potenza n-ESIMA intera di un numero Calcolare la potenza con un procedimento ricorsivo.

La potenza x^n è uguale a $x*x*....*x$, cioè si ottiene moltiplicando n volte il numero c per se stesso.

Usando un procedimento ricorsivo, si può dire che:

- $x^0 = 1$
- $x^n = x * x^{n-1}$

per esempio $8^5 = 8 * 8^4$.

Programma Python(PotenzaRicorsiva.py)

```
potenza ricorsiva.py - C:/Users/ANNA/Desktop/ex Python/potenza ricorsiva.py (3.9.1)
File Edit Format Run Options Window Help
# PotenzaRicorsiva.py: calcolo della potenza intera con funzione ricorsiva
# funzione ricorsiva per il calcolo della potenza
def potenza(x, n):
    if n == 0:
        p = 1
    else:
        p = x * potenza(x, n-1)
    return p
# funzione principale
def main():
    base = int(input("Base: "))
    esponente = int(input("Esponente: "))
    print("Potenza =", potenza(base, esponente))

# istruzione di avvio del programma
main()
```

La rappresentazione in base 2 di un numero intero Convertire un numero decimale in binario.

Per la rappresentazione in base 2 di un numero intero si deve procedere a successive divisioni intere del numero, fino ad ottenere 0 come dividendo: la successione dei resti delle divisioni, presi nell'ordine inverso rispetto a come sono stati creati, rappresenta il numero in base 2.

L'utilizzo ricorsivo di una funzione, che divida il numero e comunichi il resto della divisione, permette di ottenere la successione dei resti in ordine inverso.

Il programma contiene una struttura while per controllare che l'utente inserisca un numero positivo: in caso contrario la ripetizione richiede un nuovo valore.

Programma Python (ConvertiBinario.py)

```
conversione base 2 ricorsiva.py - C:/Users/ANNA/Desktop/ex Python/conversione base 2 ric...
File Edit Format Run Options Window Help
# ConvertiBinario.py: rappresentazione in base 2 di un numero
# funzione ricorsiva per la conversione in binario
def binario(x):
    if x != 0:
        binario(x // 2)
        print(x % 2, end = ' ')
# funzione principale
def main():
    n = 0
    while n <= 0:
        n = int(input("Numero da convertire in binario: "))
        binario(n)
# istruzione di avvio del programma
main()
```

L'istruzione PASS

L'istruzione **pass** indica all'interprete Python che non deve eseguire nessuna istruzione in una riga del programma. Essa viene usata quando un'istruzione è sintatticamente richiesta, ma il programma non deve compiere alcuna azione.

Esempio

Nella struttura *tryexcept* per la gestione delle eccezioni si può scrivere *pass* come istruzione per ignorare uno specifico tipo di eccezione:

try:

istruzioni1

except ValueError:

pass

n.b.:

un secondo uso pratico, molto comune, dell'istruzione *pass* riguarda la possibilità di lasciare in sospeso la definizione di una funzione , o in generale di un blocco di codice, in **fase di sviluppo del software**.

def f1(x):

pass

in questo caso la parola chiave *pass* svolge il ruolo di segnaposto per le istruzioni che saranno inserite successivamente .

Sempre in fase di **collaudo** di un programma, è utile testare l'esecuzione delle istruzioni di elaborazione e di calcolo ignorando le istruzioni di output formattato, nel caso siano complesse ed in numero elevato.

Esempio

def f2(x):

calcolo1(y, z)

calcolo2(u, w)

pass

In una struttura *for* ci può essere l'esigenza di saltare qualche passo della ripetizione. In questo caso, l'istruzione *pass* è equivalente all'istruzione *continue* (vedi pag.)

Esempio

for i **in** range(n):

if i == 0:

pass

else:

.....

I moduli

Come già accennato in precedenza (vedi pag. 41), il **modulo** è un file con estensione *.py*, memorizzato su disco e contenente definizioni (per esempio definizioni di funzioni), istruzioni e costanti.

I moduli nella programmazione nel linguaggio Python rispondono a due importanti esigenze:

- suddividere un programma complesso in diversi file per facilitare il controllo e la manutenzione del software;
- salvare all'interno di un file una o più funzioni che possono essere utili in programmi diversi, senza la necessità di copiare ogni volta la loro definizione nel codice del programma.

Il contenuto di un modulo diventa disponibile in un programma tramite l'istruzione **import**.

Esempio

In alcuni degli esempi visti in precedenza è stata usata l'istruzione *math* per importare la libreria delle funzioni matematiche:

import math

In effetti *math* è un esempio di **modulo predefinito** del linguaggio.

Dopo l'importazione, le funzioni e gli identificativi definiti nel modulo sono richiamabili nel programma con la **notazione con il punto**:

nomemodulo.nomefunzione
nomemodulo.nomeidentificativo

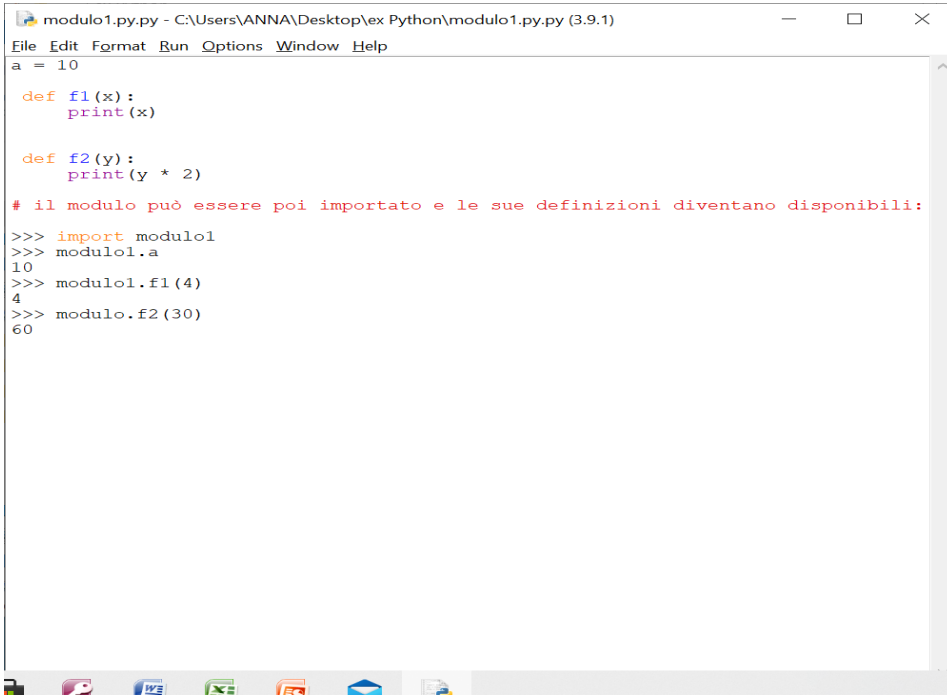
In generale:

nomemodulo.nomerisorsa

Il nome del modulo è il **qualificatore** della risorsa disponibile nel modulo ed usata nel programma.

Esempio

Il seguente file (*modulo1.py*) contiene alcune definizioni e viene salvato su disco come un modulo:



```
modulo1.py - C:\Users\ANNA\Desktop\ex Python\modulo1.py (3.9.1)
File Edit Format Run Options Window Help
a = 10

def f1(x):
    print(x)

def f2(y):
    print(y * 2)

# il modulo può essere poi importato e le sue definizioni diventano disponibili:
>>> import modulo1
>>> modulo1.a
10
>>> modulo1.f1(4)
4
>>> modulo1.f2(30)
60
```

Se necessario, il modulo può essere rimosso dalla MC con il comando **del**, che rappresenta l'operazione opposta all'importazione :

	<p>>>> del modulo1</p> <p>n.b.: Nella fase di sviluppo e collaudo del software, può accadere di dover modificare il modulo e poi ricaricare il modulo stesso. Per questo si usano i comandi:</p> <pre>>>> import importlib >>> importlib.reload(modulo1)</pre> <p>Con il primo comando viene importata la libreria importlib che gestisce l'<i>import</i> di Python. Il secondo comando ricarica il modulo con la funzione reload di <i>importlib</i>.</p> <p>Un modulo può contenere, oltre alle definizioni di funzioni e identificativi, anche istruzioni esecutive. In questo caso le istruzioni, scritte nel modulo all'esterno delle definizioni, vengono mandate in esecuzione quando il programma incontra il comando <i>import</i>.</p> <p>Esempio Negli esempi precedenti sono state inserite nel programma le definizioni delle funzioni di elaborazione e della funzione <i>main()</i> per indicare la funzione principale del programma. È stata anche inserita l'istruzione <i>main()</i> come chiamata alla funzione <i>main()</i> e come avvio del programma. Se si considera lo script come modulo, il comando <i>import</i> avvia l'esecuzione dello script, perché viene eseguita l'istruzione <i>main()</i> scritta all'esterno delle definizioni.</p> <p>Un modulo può, a sua volta, importare altri moduli con il comando <i>import</i>.</p>
<p>La clausola <i>from</i></p>	<p>Ogni modulo possiede una propria tabella di nomi. Nell'operazione di importazione, possono essere caricate anche singole risorse del modulo indicandone il nome con il comando from...import. Se le risorse sono più di una, i loro nomi sono scritti uno di seguito all'altro dopo <i>import</i> e separati dalla virgola.</p> <p>Se si usa la clausola <i>from</i>, non è necessario specificare il nome del modulo quando si usano le funzioni o gli identificativi importati.</p> <p>Le funzioni che non sono state importate non sono disponibili.</p> <p>Esempio</p> <pre>>>> from modulo1 import a, f1 >>> a 10 >>> f1(5) 5 >>> f2(8) NameError: name 'f2' is not defined</pre> <p>Per importare tutti i nomi definiti in un modulo, si può scrivere il comando con l'asterisco * (che significa tutti):</p> <pre>>>> from modulo1 import *</pre>

	<p>Il comando precedente è equivalente a <i>import modulo1</i>, rispetto al quale offre il vantaggio di poter tralasciare il nome del modulo come qualificatore dei nomi delle risorse.</p> <p>Tuttavia l'uso del qualificatore spesso è preferibile perché rende più leggibile il codice ed evita ambiguità nel caso di funzioni diverse (o identificativi) aventi lo stesso nome: si pensi alla funzione predefinita <i>pow()</i> e alla funzione <i>math.pow()</i> viste in precedenza. Di contro l'importazione con <i>import *</i> risparmia molte battute da tastiera nelle attività svolte in modalità interattiva.</p> <p>n.b.:</p> <p>il comando from ... import *, importa tutti i nomi di un modulo ad eccezione di quelli che iniziano con il carattere _ (underscore) che, per convenzione, identifica le variabili non pubbliche di un modulo.</p>
La clausola <i>as</i>	<p>Per documentare meglio il codice di un programma, o per usare nomi più corti come riferimento ai moduli importanti, si possono usare alias per i nomi dei moduli o delle funzioni importate.</p> <p>L'<i>alias</i> è definito con la clausola as.</p> <p>Esempio</p> <pre>>>> import modulo1 as m1 >>> m1.f1(3) 3 >>> from modulo1 import f2 as doppio >>> doppio(12) 24</pre>
La funzione predefinita <i>dir()</i>	<p>La funzione <i>dir()</i> applicata ad un modulo restituisce l'elenco dei nomi in esso definiti.</p> <p>Esempio</p> <p>Con i seguenti comandi si ottiene l'elenco dei nomi definiti nel modulo <i>utente</i> creato dall'utente:</p> <pre>>>> import modulo1 >>> dir (modulo1) ['_builtins_', '_cached_', '_doc_', '_file_', '_loader_',]</pre> <p>Con comandi analoghi si può ottenere l'elenco dei nomi di un modulo predefinito del linguaggio (in pratica i nomi delle funzioni matematiche disponibili nella libreria <i>math</i>):</p> <pre>>>> import math >>> dir (math) ['_doc_', '_loader_', '_name_', '_package_', '_spec_', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt', 'lcm', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'nextafter', 'perm', 'pi', 'pow', 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc', 'ulp']</pre>

<p>Negli output degli esempi precedenti compaiono dei nomi con doppio underscore all'inizio e in coda: essi rappresentano i nomi riservati di funzioni o variabili predefinite (dette anche funzioni o variabili <i>globali speciali</i>). L'uso del <i>doppio underscore</i> previene i conflitti con eventuali nomi uguali definiti dall'utente.</p> <p>Esempio</p> <p>La tabella mostra alcuni esempi di uso del doppio underscore:</p> <table> <tr> <td><code>_builtins_</code></td><td>Gli oggetti predefiniti di Python</td></tr> <tr> <td><code>_cached_</code></td><td>Il percorso di registrazione della versione compilata del modulo</td></tr> <tr> <td><code>_doc_</code></td><td>La docstring di documentazione del modulo</td></tr> <tr> <td><code>_file_</code></td><td>Il percorso di registrazione del modulo</td></tr> <tr> <td><code>_loader_</code></td><td>L'oggetto <i>loader</i> utilizzato per caricare il modulo in memoria</td></tr> <tr> <td><code>_name_</code></td><td>Il nome del modulo stesso</td></tr> </table> <p>La docstring è una stringa racchiusa tra triple virgolette (o tripli apici) che contiene brevi testi di documentazione di moduli e funzioni.</p>		<code>_builtins_</code>	Gli oggetti predefiniti di Python	<code>_cached_</code>	Il percorso di registrazione della versione compilata del modulo	<code>_doc_</code>	La docstring di documentazione del modulo	<code>_file_</code>	Il percorso di registrazione del modulo	<code>_loader_</code>	L'oggetto <i>loader</i> utilizzato per caricare il modulo in memoria	<code>_name_</code>	Il nome del modulo stesso
<code>_builtins_</code>	Gli oggetti predefiniti di Python												
<code>_cached_</code>	Il percorso di registrazione della versione compilata del modulo												
<code>_doc_</code>	La docstring di documentazione del modulo												
<code>_file_</code>	Il percorso di registrazione del modulo												
<code>_loader_</code>	L'oggetto <i>loader</i> utilizzato per caricare il modulo in memoria												
<code>_name_</code>	Il nome del modulo stesso												

<p>La funzione predefinita <i>help()</i></p>	<p>La funzione <code>help()</code> fornisce informazioni sul modulo indicato come argomento.</p> <p>Esempio</p> <pre>>>> import modulo1 >>> help(modulo1) Help on module modulo1:</pre> <p>NAME</p> <p> modulo1</p> <p>FUNCTIONS</p> <p> f1(x)</p> <p> f2(y)</p> <p>DATA</p> <p> a = 10</p> <p>FILE</p> <p> c:\esercizipython\modulo1.py</p> <p>Le informazioni di <i>help()</i> possono riguardare singole funzioni.</p> <p>Esempio</p> <pre>>>> import modulo1 >>> help(modulo1.f2)</pre>
---	--

	<p>Help on function f2 in module modulo1:</p> <p>f2(y)</p> <pre>>>> import math >>> help (math.sqrt)</pre> <p>Help on built-in function sqrt in module math:</p> <p>sqrt(x, /)</p> <p> return the square root of x.</p>												
Coding	<p>La bolletta dell'acqua Fatturare annualmente i consumi dell'acqua.</p> <p>Il problema della fatturazione dei consumi riguarda le aziende di erogazione di servizi pubblici in generale e, in particolare, le società che gestiscono il servizio idrico (acqua) e i fornitori di energia elettrica e gas.</p> <p>La fatturazione è di solito periodica, con emissione di una bolletta di pagamento ogni 2 o 3 mesi, oppure anche con una sola bolletta annuale, in alcuni casi, per i consumi di acqua.</p> <p>In questo caso si prende in esame la fatturazione dei consumi di acqua, nell'ipotesi semplificativa di una sola fattura annuale.</p> <p>Per i consumi di acqua l'unità di misura è il metro cubo (<i>mc</i>).</p> <p>Le tariffe sono normalmente distinte tra Privati (uso domestico o clienti residenziali) e Aziende (uso diverso dal domestico).</p> <p>I valori sono immessi dall'utente all'inizio del programma.</p> <p>Per le aziende il calcolo delle tariffe è semplice: si moltiplica il consumo in mc per il prezzo unitario al mc (1,25 €).</p> <p>Per i privati si suppone che esistano tre fasce di prezzo in base al consumo annuale, secondo la seguente tabella:</p> <table><tr><th>Tipo tariffa</th><th>Consumo annuo in mc</th><th>Prezzo unitario al mc</th></tr><tr><td>Agevolata</td><td>fino a 80</td><td>0,31 €</td></tr><tr><td>Base</td><td>da 81 a 120</td><td>0,61 €</td></tr><tr><td>Eccedenza</td><td>oltre 120</td><td>1,09 €</td></tr></table> <p>Il calcolo è fatto a scaglioni: sui primo 80 mc si paga 3,31, sui consumi oltre gli 80 mc si paga 0,61€ etc.</p> <p>Per esempio, se il consumo annuo è stato di 90 mc, il costo sarà: (80 * 0,31) + (10 * 0,61) = 30,90€</p> <p>Sia per l'azienda sia per il privato, occorre aggiungere al risultato del calcolo dei consumi effettivi (<i>quota variabile</i>) una <i>quota fissa annuale</i>, che è indipendente dai consumi:</p> <ul style="list-style-type: none">• per le aziende 21€;• per i privati 14€. <p>Infine occorre applicare l'imposta IVA, secondo una percentuale che può essere considerata una costante, per esempio il 10%.</p> <p>Nel caso generale di fatturazione periodica (non annuale), nel procedimento di calcolo si dovrebbe considerare, oltre al consumo annuo, anche il consumo del periodo e</p>	Tipo tariffa	Consumo annuo in mc	Prezzo unitario al mc	Agevolata	fino a 80	0,31 €	Base	da 81 a 120	0,61 €	Eccedenza	oltre 120	1,09 €
Tipo tariffa	Consumo annuo in mc	Prezzo unitario al mc											
Agevolata	fino a 80	0,31 €											
Base	da 81 a 120	0,61 €											
Eccedenza	oltre 120	1,09 €											

dividere la quota fissa annuale per il numero di mesi compresi dal periodo di fatturazione.

Poiché il problema è complesso, si procede con il metodo di sviluppo top-down, individuando i seguenti sottoproblemi:

- acquisizione dei dati sulle tariffe;
- calcolo del totale da pagare (quota variabile + quota fissa+ IVA);
- visualizzazione dei risultati.

Il secondo punto può essere ulteriormente scomposto in due sottoproblemi:

- calcolo per aziende;
- calcolo per privati.
-

Input: consumo, mc
Output: qv, qf, iva, totale

Legenda:
mc = metro cubo
qv = quota variabile
qf = quota fissa
totale = totale da pagare

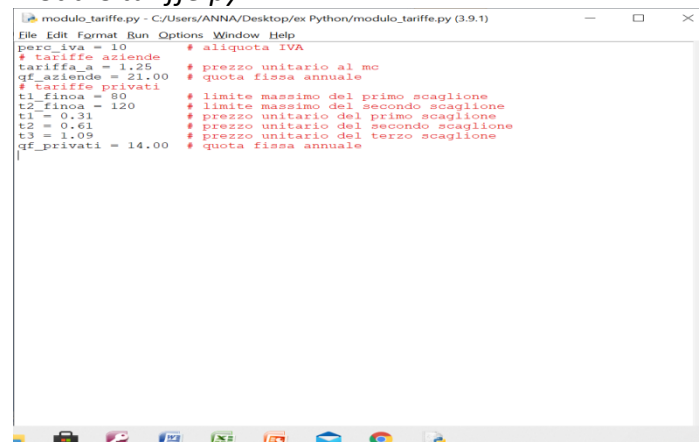
Modulo delle tariffe

Il modulo delle tariffe (*tariffe.py*) contiene un elenco di costanti (con i nomi tutti maiuscoli) da utilizzare per i calcoli:

- *PERC_IVA*, cioè l'aliquota IVA;
- **Tariffe per le aziende:**
TARIFFA_A, prezzo unitario al mc;
QF_AZIENDE, quota fissa annuale;
- **Tariffe per i privati:**
T1_FINOA, limite max del primo scaglione;
T2_FINOA, limite max del secondo scaglione;
T1, prezzo unitario del primo scaglione;
T2, prezzo unitario del secondo scaglione;
T3, prezzo unitario del terzo scaglione;
QF_PRIVATI, quota fissa annuale.

Il limite minimo del primo scaglione è 0, mentre i limiti degli altri scaglioni si ottengono aggiungendo 1 al valore massimo dello scaglione precedente.

Modulo *tariffe.py*



```
modulo_tariffe.py - C:/Users/ANNA/Desktop/ex Python/modulo_tariffe.py (3.9.1)
File Edit Format Run Options Window Help
perc_iva = 10 # aliquota IVA
# tariffe aziende
tariffa_a = 1.25 # prezzo unitario al mc
qf_aziende = 21.00 # quota fissa annuale
# tariffe privati
t1_finoa = 50 # limite massimo del primo scaglione
t2_finoa = 120 # limite massimo del secondo scaglione
t1 = 0.31 # prezzo unitario del primo scaglione
t2 = 0.61 # prezzo unitario del secondo scaglione
t3 = 1.09 # prezzo unitario del terzo scaglione
qf_privati = 14.00 # quota fissa annuale
```

n.b.:

L'uso di un modulo contenente delle costanti per memorizzare i dati di input offre il vantaggio di poter modificare nel tempo i valori per la fatturazione. Il contenuto del modulo può così essere modificato con un qualsiasi editor di testo, anche da parte dell'utente del programma senza l'intervento del programmatore.

Modulo di calcolo

Il modulo per il calcolo (*calcoli.py*) contiene due funzioni che ricevono come argomento il consumo in mc: *calcolo_aziende()* e *calcolo_privati()*. Una terza funzione *risultati()* si occupa della visualizzazione dei valori ottenuti dalle funzioni.

All'inizio il modulo di calcolo importa tutti i nomi delle costanti del modulo delle tariffe.

Modulo(*calcoli.py*)

```
calcoli.py - C:\Users\ANNA\Desktop\ex Python\ex con uso di moduli\calcoli.py (3.9.1)
File Edit Format Run Options Window Help
# importazione delle tariffe
from modulo_tariffe import *

# calcolo della bolletta con le tariffe per le aziende
def calcolo_aziende(c):
    qv = c * tariffa_a                # quota variabile
    qf = qf_aziende                  # quota fissa
    iva = (qf + qv) * perc_iva / 100  # imposta
    totale = qf + qv + iva           # totale
    risultati(qv, qf, iva, totale)

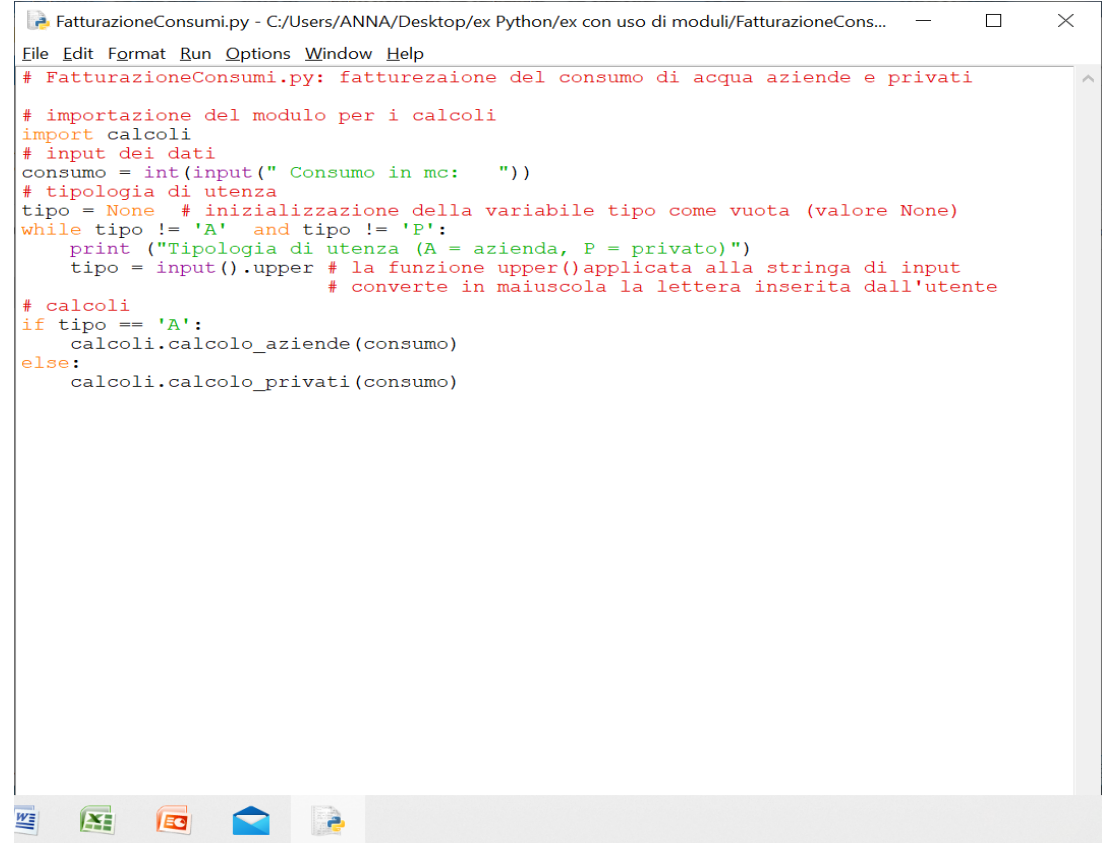
# calcolo della bolletta con le tariffe per i privati
def calcolo_privati(c):
    # quota variabile
    if c <= t1_finoa:
        qv = c * t1
    elif c <= t2_finoa:
        qv = (t1_finoa * t1) + (c - t1_finoa) * t2
    else:
        qv = (t1_finoa * t1) + (t2_finoa - t1_finoa) * t2 + \
            (c - t2_finoa) * t3
    qf = qf_privati
    iva = (qf + qv) * perc_iva / 100  # imposta
    totale = qf + qv + iva           # totale
    risultati(qv, qf, iva, totale)

# output dei risultati
def risultati (qv, qf, iva, totale):
    print('quota variabile: {: 10.2f}'.format(qv))
    print('quota fissa:      {: 10.2f}'.format(qf))
    print('IVA:              {: 10.2f}'.format(iva))
    print('Totale:          {: 10.2f}'.format(totale))
```

Il programma principale (*FatturazioneConsumi.py*) importa il modulo dei calcoli e richiede da tastiera il valore del consumo in mc e la tipologia di utenza (azienda o privato): la tipologia di utenza determina la chiamata della funzione *calcolo_aziende()* oppure della funzione *calcolo_privati()*.

Alla richiesta della tipologia, con una struttura *while* viene effettuato un controllo sull'input dell'utente, in modo da impedire l'input di una lettera che non sia né A né P. Infine la chiamata alla funzione *risultati()* produce la visualizzazione dei valori restituiti dalla funzione di calcolo.

Programma Python (FatturazioneConsumi.py)



```
FatturazioneConsumi.py - C:/Users/ANNA/Desktop/ex Python/ex con uso di moduli/FatturazioneCons...
File Edit Format Run Options Window Help
# FatturazioneConsumi.py: fatturezaione del consumo di acqua aziende e privati

# importazione del modulo per i calcoli
import calcoli
# input dei dati
consumo = int(input(" Consumo in mc:  "))
# tipologia di utenza
tipo = None # inizializzazione della variabile tipo come vuota (valore None)
while tipo != 'A' and tipo != 'P':
    print ("Tipologia di utenza (A = azienda, P = privato)")
    tipo = input().upper # la funzione upper() applicata alla stringa di input
                        # converte in maiuscola la lettera inserita dall'utente

# calcoli
if tipo == 'A':
    calcoli.calcolo_aziende(consumo)
else:
    calcoli.calcolo_privati(consumo)
```

I moduli predefiniti

Oltre ai moduli definiti dal programmatore, in un programma si possono usare i **moduli predefiniti** (*built-in*) nel linguaggio Python, come il modulo **math** in precedenza.

I moduli predefiniti appartengono alla **libreria standard** di Python che comprende funzioni, tipi, costanti di uso comune nella programmazione.

L'importazione di un modulo predefinito può essere completa oppure selettiva, cioè che riguarda solo alcune funzioni.

Esempio

```
import math # importa tutto il modulo
```

```
from math import sqrt, pi # importa solo la funzione sqrt e la costante pi
```

Anche per i moduli predefiniti si possono usare le funzioni già viste: la funzione **dir()** per ottenere l'elenco delle funzioni definite in precedenza e la funzione **help()** per le informazioni sul modulo o su una singola funzione.

Esempio

```
import math # importa tutto il modulo
```

```
from math import sqrt, pi # importa solo la funzione sqrt e la costante pi
```

Anche per i moduli predefiniti si possono usare le funzioni viste in precedenza: la funzione **dir()** per ottenere l'elenco delle funzioni definite in precedenza e la funzione **help()** per le informazioni sul modulo o su una singola funzione.

Il modulo **sys**

Il modulo **sys** mette a disposizione le variabili e le funzioni che interagiscono direttamente con l'interprete. Per usare le risorse del modulo, occorre prima importarlo:

```
>>> import sys
```

La variabile **version** contiene il numero di versione dell'interprete Python in uso:

```
>>> sys.version
```

La variabile **path** contiene i percorsi impostati per la ricerca dei moduli o degli script di Python:

```
>>> import sys
>>> sys.path
```

Per aggiungere un nuovo percorso si usa la funzione **append**:

```
>>> sys.path.append( 'C:\\EserciziPython' )
```

Per eliminare un percorso di ricerca si usa la funzione **remove**:

```
>>> sys.path.remove( 'C:\\EserciziPython' )
```

Le modifiche ai percorsi di ricerca si possono verificare con il comando:

```
>>> sys.path
```

n.b.:

L'aggiunta di un percorso rende disponibili i moduli direttamente dal prompt della Shell di IDLE con l'operazione di importazione.

La variabile **arg** contiene la lista degli argomenti passati allo script quando viene eseguito dalla linea comandi. In particolare: l'elemento della lista di indice 0, *arg[0]*, è il nome dello script, *arg[1]* è il primo argomento, *arg[2]* è il secondo etc.

Per conoscere il numero di argomenti si usa la funzione **len()** applicata alla lista.

Esempio:

Il seguente script *CalcolaPerimetro.py* calcola il perimetro del quadrato avente il lato fornito come argomento nell'esecuzione dalla linea comandi:



```
CalcolaPerimetro.py - C:\Users\ANNA\Desktop\ex Python\ex con uso di moduli\CalcolaPerim...
File Edit Format Run Options Window Help
# perimetro del quadrato
import sys
lato = int(sys.argv[1])
print("perimetro =", lato * 4)

Ln: 1 Col: 0
```

Eseguendo lo script dalla linea di comandi, il valore del lato fornito da tastiera, insieme al comando, viene associato all'elemento di indice 1 della lista *argv*.

`python CalcolaPerimetro.py 10`

L'output del comando è: `perimetro = 40`

Il modulo **os**

Il modulo **os** mette a disposizione molte funzioni per interagire con il sistema operativo . E' un modulo importante perché fornisce la **portabilità** del codice Python nelle funzionalità che dipendono dal SO in uso.

Per usare le funzioni di questo modulo, occorre prima importarlo:

```
>>> import os
```

La funzione **os.getcwd()** restituisce la directory corrente, mentre la funzione **os.chdir()** cambia la directory corrente con quella specificata come argomento.

La funzione **os.listdir()** fornisce l'elenco dei nomi dei file nella directory corrente.

La funzione **os.mkdir()** crea una nuova directory.

La funzione **os.rename()** cambia il nome di un file o di una directory.

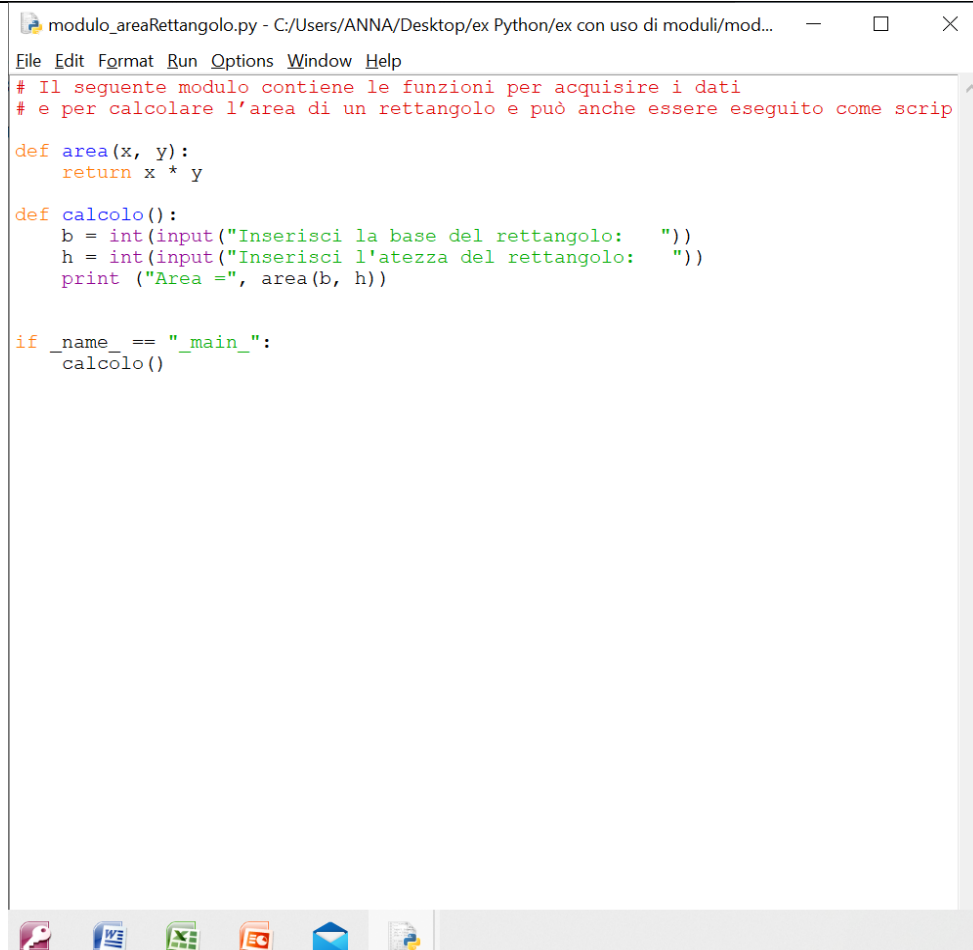
La funzione **os.system()** esegue un comando della shell del SO in uso. Il comando è specificato come argomento della funzione. La funzione restituisce il valore 0 se il comando è stato eseguito correttamente, oppure il valore 1 (oppure diverso da 0) se sono stati riscontrati errori nell'esecuzione.

Esempio:

l'elenco dei nomi dei file nella directory corrente, scritta su un file con una ridirezione dell'output, si ottiene:

- nel SO Windows con:

	<pre>>>> os.system ('dir > elenco')</pre> <ul style="list-style-type: none"> • nel SO Linux con: <pre>>>> os.system ('ls > elenco')</pre>
L'esecuzione dei moduli come script a sé stanti	<p>Lo scopo principale del modulo è di rendere disponibili le funzioni in esso contenute, tramite l'importazione in un programma. Tuttavia spesso si ha la necessità di testare una o più funzioni di un modulo senza importarle nel programma.</p> <p>Quando l'interprete Python avvia l'esecuzione di un modulo (o uno script) assegna all'attributo <code>_name_</code> del modulo la stringa predefinita <code>_main_</code>. Questo consente di controllare se il modulo è stato avviato in esecuzione come script a sé stante, con una struttura <i>if</i> come la seguente:</p> <pre>if _name_ == "_main_" :</pre> <p>Esempio: Supponiamo di avere un modulo che contiene due funzioni f1 e f2, disponibili per l'importazione:</p> <pre>def f1(): print ("funzione 1") def f2(): print ("funzione 2")</pre> <p>Aggiungendo le seguenti tre righe di codice alla fine del modulo, si può richiedere l'esecuzione delle funzioni del modulo per scopi di collaudo, come se fosse uno script:</p> <pre>if _name_ == "_main_" : f1() f2()</pre> <p>Con questa modifica, il modulo può comunque essere importato in un programma, ma può anche essere eseguito come script a sé stante.</p> <p>Il seguente modulo contiene le funzioni per acquisire i dati e per calcolare l'area di un rettangolo e può anche essere eseguito come script:</p>



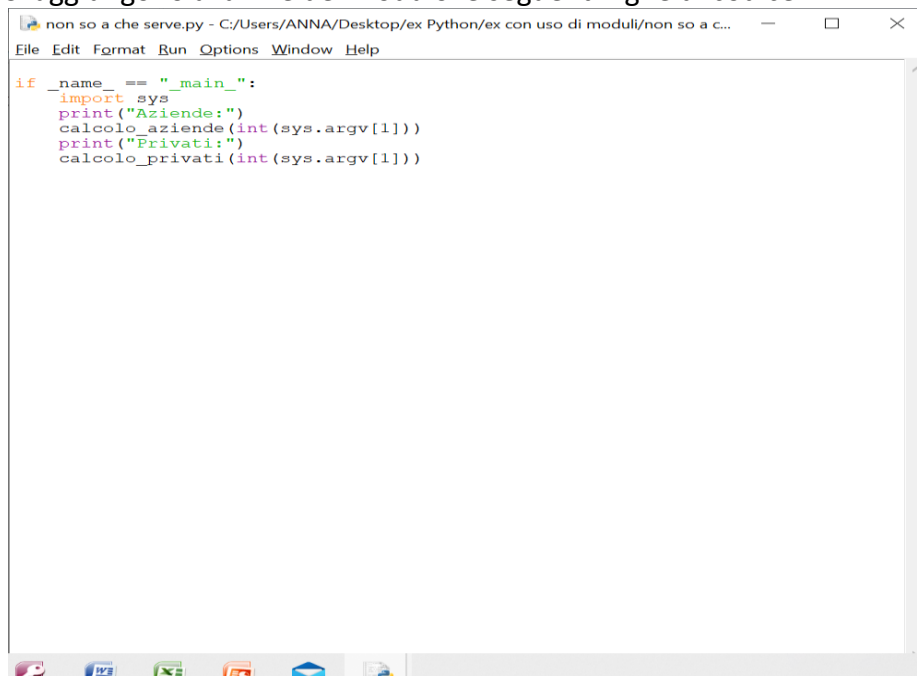
```
modulo_areaRettangolo.py - C:/Users/ANNA/Desktop/ex Python/ex con uso di moduli/mod...
File Edit Format Run Options Window Help
# Il seguente modulo contiene le funzioni per acquisire i dati
# e per calcolare l'area di un rettangolo e può anche essere eseguito come scrip

def area(x, y):
    return x * y

def calcolo():
    b = int(input("Inserisci la base del rettangolo: "))
    h = int(input("Inserisci l'atezza del rettangolo: "))
    print ("Area =", area(b, h))

if __name__ == "__main__":
    calcolo()
```

Nel coding per il calcolo della bolletta dell'acqua, il modulo *calcoli.py* contiene le funzioni di calcolo richiamate dal programma principale *Fatturazione Consumi.py*. Si aggiungono alla fine del modulo le seguenti righe di codice:

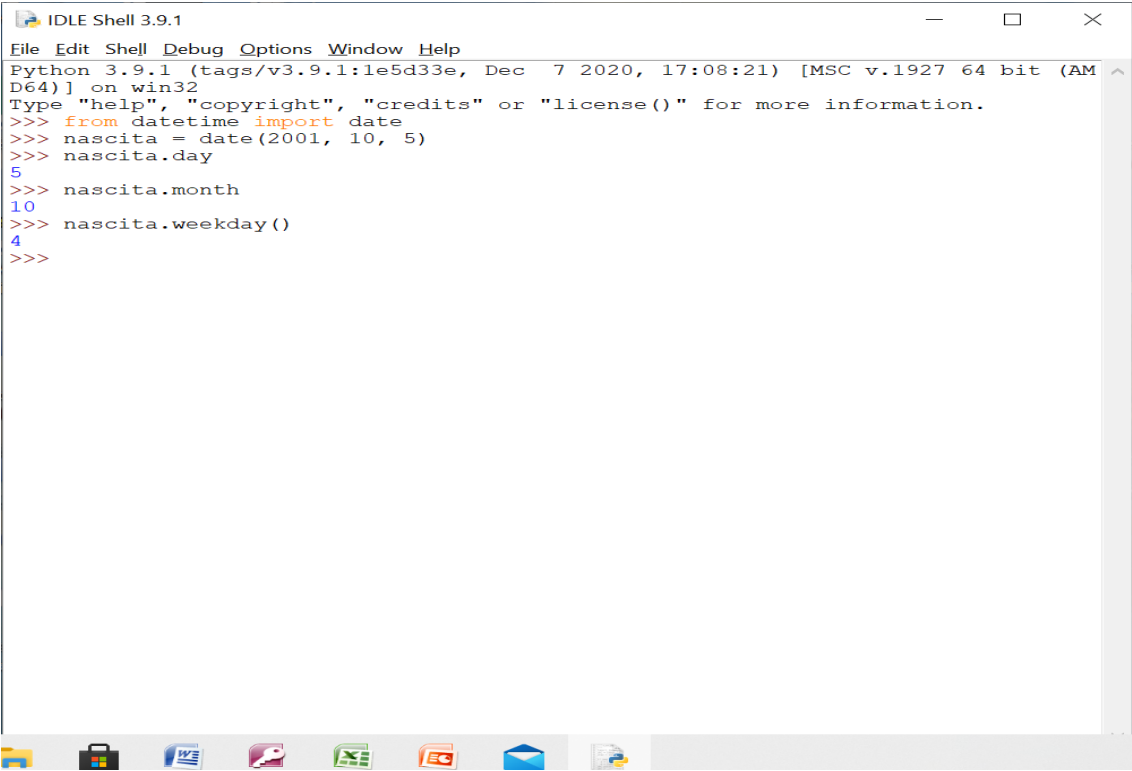


```
non so a che serve.py - C:/Users/ANNA/Desktop/ex Python/ex con uso di moduli/non so a c...
File Edit Format Run Options Window Help

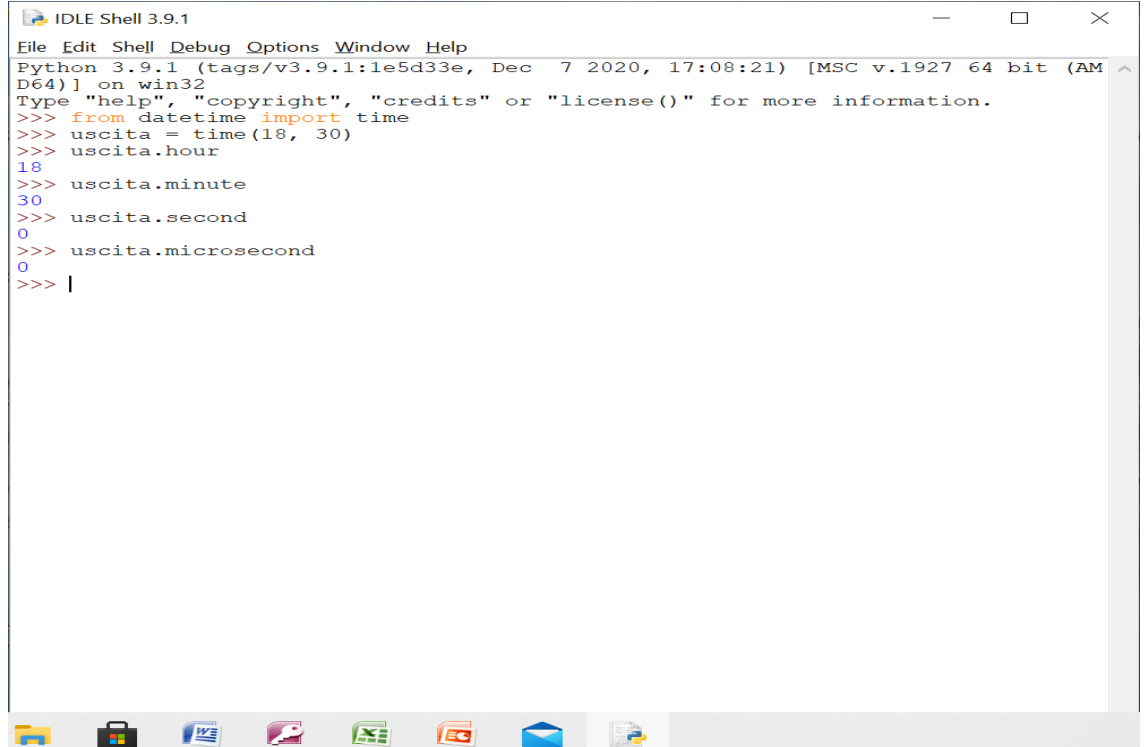
if __name__ == "__main__":
    import sys
    print("Aziende:")
    calcolo_azienze(int(sys.argv[1]))
    print("Privati:")
    calcolo_privati(int(sys.argv[1]))
```

Le funzioni *calcolo_azienze* e *calcolo_privati* possono essere collaudate eseguendo il modulo direttamente dalla linea comandi e fornendo come argomento il valore del consumo in mc: `python calcoli.py 90`

Lo stesso modulo, modificato come sopra, può essere ancora importato, perché nel

	<p>caso dell'importazione la condizione <code>if_name_ == "_main_"</code> risulta falsa e le istruzioni dopo <code>if</code> non vengono eseguite.</p>
<p>Le date e gli orari</p>	<p>Il modulo datetime del linguaggio Python mette a disposizione gli oggetti e le funzioni (<i>metodi delle classi</i>) per definire e manipolare date e orari.</p> <p>Le classi principali che consentono di utilizzare oggetti di tipo data e orario sono:</p> <ul style="list-style-type: none"> • date: rappresenta una data; • time: rappresenta un orario; • datetime: rappresenta una combinazione di data e orario; • timedelta: rappresenta una durata come differenza tra due oggetti di tipo <i>date</i>, <i>time</i> o <i>datetime</i>. <p>Per definire un oggetto di tipo date, occorre fornire, nell'ordine, i valori agli attributi: anno (year), mese (month) e giorno (day).</p> <p>Il metodo weekday() applicato ad un oggetto di tipo <i>date</i> restituisce il numero del giorno della settimana (0= lunedì, 6= domenica).</p> <p>Esempio:</p>  <pre> Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32 Type "help", "copyright", "credits" or "license()" for more information. >>> from datetime import date >>> nascita = date(2001, 10, 5) >>> nascita.day 5 >>> nascita.month 10 >>> nascita.weekday() 4 >>> </pre> <p>In modo analogo si può definire un oggetto di tipo <i>time</i> che ha come attributi: le ore da 0 a 23 (hour), i minuti da 0 a 59 (minute), i secondi da 0 a 59 (second), i microsecondi da 0 a 1000000 (microsecond).</p> <p>Nella definizione di un orario tutti gli argomenti sono opzionali (gli attributi non definiti hanno il valore di <i>default</i> uguale a 0).</p>

Esempio:



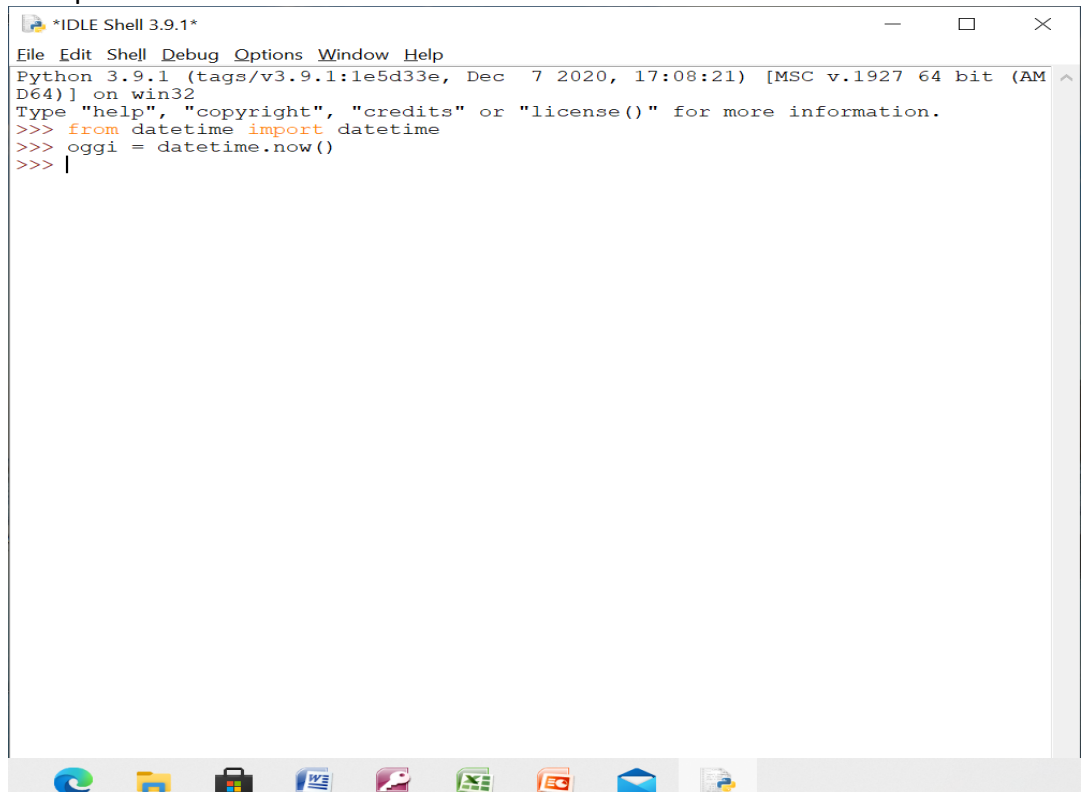
```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> from datetime import time
>>> uscita = time(18, 30)
>>> uscita.hour
18
>>> uscita.minute
30
>>> uscita.second
0
>>> uscita.microsecond
0
>>> |
```

Il metodo **today()** di un oggetto di tipo *date* restituisce la data corrente.

```
>>> oggi = date.today()
```

Se si vuole ottenere la data e l'orario corrente, occorre usare un oggetto di tipo **datetime** ed il metodo **now()**.

Esempio:



```
*IDLE Shell 3.9.1*
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> from datetime import datetime
>>> oggi = datetime.now()
>>> |
```

Anche l'oggetto *datetime* possiede gli attributi *year*, *month*, *day*, *hour*, *minute*, *second*, *microsecond* e il metodo *weekday()*.

<p>La distanza tra due date</p>	<p>La distanza tra due date è un oggetto di tipo timedelta e il suo attributo days fornisce il valore in giorni.</p> <p>Esempio:</p>  <pre> Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32 Type "help", "copyright", "credits" or "license()" for more information. >>> from datetime import date >>> data1 = date(2001, 10, 5) >>> data2 = date(2003, 10, 5) >>> distanza = data2 - data1 >>> distanza datetime.timedelta(days=730) >>> distanza.days 730 >>> </pre> <p>Se si importa anche la classe timedelta si possono definire le durate, espresse in giorni (days), secondi (second), microsecondi (microseconds), minuti (minutes), ore (hours) e settimane (weeks), per calcolare incrementi di date ed orari.</p> <p>Esempio:</p> <pre> >>> from datetime import timedelta >>> data3 = data2 + timedelta(days = 730) >>> data3 datetime.date(2005, 10, 4) >>> data4 = data3 +timedelta(weeks = 3) >>> data4 datetime.date(2005, 10, 25) </pre>
<p>L'output formattato delle date</p>	<p>Il metodo strftime(), applicato a oggetti di tipo <i>date</i> e <i>datetime</i>, formatta l'output di una data usando una stringa di formattazione nella quale sono inserite delle marcature che indicano giorno, mese, anno, nome del giorno e del mese. La stringa di formattazione è riportata esattamente come è scritta, sostituendo alle marcature il corrispondente valore.</p> <p>L'elenco delle marcature è riportato nella seguente tabella:</p>

	<table><tr><th>marcatura</th><th>Esempio (venerdì 5 ottobre 2001)</th><th>Contenuto della stringa</th></tr><tr><td>%d</td><td>'05'</td><td>Giorno del mese</td></tr><tr><td>%m</td><td>'10'</td><td>Numero del mese</td></tr><tr><td>%y</td><td>"01</td><td>Anno con due cifre</td></tr><tr><td>%Y</td><td>'2001'</td><td>Anno con 4 cifre</td></tr><tr><td>%a</td><td>'Fri'</td><td>Nome del giorno abbreviato</td></tr><tr><td>%A</td><td>'Friday'</td><td>Nome del giorno</td></tr><tr><td>%b</td><td>'Oct'</td><td>Nome del mese abbreviato</td></tr><tr><td>%B</td><td>'October'</td><td>Nome del mese</td></tr></table> <p>Esempio:</p> <pre>>>> nascita = date(1979, 3, 14) >>> print (nascita.strftime("%d-%m-%y")) 14-03-79 >>> print (nascita.strftime("%A %d %B %Y")) Wednesday 14 March 1979</pre>	marcatura	Esempio (venerdì 5 ottobre 2001)	Contenuto della stringa	%d	'05'	Giorno del mese	%m	'10'	Numero del mese	%y	"01	Anno con due cifre	%Y	'2001'	Anno con 4 cifre	%a	'Fri'	Nome del giorno abbreviato	%A	'Friday'	Nome del giorno	%b	'Oct'	Nome del mese abbreviato	%B	'October'	Nome del mese
marcatura	Esempio (venerdì 5 ottobre 2001)	Contenuto della stringa																										
%d	'05'	Giorno del mese																										
%m	'10'	Numero del mese																										
%y	"01	Anno con due cifre																										
%Y	'2001'	Anno con 4 cifre																										
%a	'Fri'	Nome del giorno abbreviato																										
%A	'Friday'	Nome del giorno																										
%b	'Oct'	Nome del mese abbreviato																										
%B	'October'	Nome del mese																										
I servizi di geolocalizzazione	<p>Nell’output delle date occorre convertire i nomi dei giorni della settimana e dei mesi in italiano. Per far questo si deve importare il modulo locale che fornisce i servizi di localizzazione, in particolare per le date, la valuta e la rappresentazione dei numeri con il punto decimale e il separatore delle migliaia.</p> <p>Con la funzione setlocale() il programma acquisisce le impostazioni del sistema in uso da parte dell’utente.</p> <p>Con le seguenti istruzioni vengono caricate tutte le impostazioni locali:</p> <pre>>>> import locale >>> locale.setlocale(locale.LC_ALL, ' ') ' Italian_Italy.1252'</pre> <p>Con queste impostazioni si possono stampare le date con i nomi in italiano.</p> <p>Esempio:</p> <pre>>>> data1 = date(2019, 12, 18) >>> print (data1.strftime("%A %d %B %Y")) mercoledì 18 dicembre 2019</pre>																											
Coding	<p>Stampare la scadenza a 30 giorni di un prodotto alimentare del quale si conosce la data di produzione.</p> <p>Il programma usa gli oggetti e i metodi visti finora per i moduli <i>datetime</i> e <i>local</i>. Viene usato anche il metodo strptime() che converte una stringa di input in un oggetto di tipo <i>datetime</i>.</p> <p>Nella conversione il metodo usa i marcatori di formattazione dell’input, come quelli del metodo strftime() per l’output.</p> <pre>giornop = input("Data di produzione (gg-mm-aaaa): ") datap = datetime.strptime (giornop, "%d-%m-%Y")</pre> <p>La data di scadenza è calcolata aggiungendo una durata di 30 giorni alla data di produzione (oggetto di tipo <i>timedelta</i>)</p>																											

Programma Python (*scadenza.py*)

```
*data di scadenza di un alimento.py - C:/Users/ANNA/Desktop/ex Python/data di scadenza ...
File Edit Format Run Options Window Help

# scadenza.py: scadenza di un prodotto alimentare

# importazione dei moduli
from datetime import datetime, timedelta
import locale

# impostazioni di localizzazione
locale.setlocale(locale.LC_ALL, '')

# programma principale
def main():
    # input della data di produzione
    giornop = input("Data di produzione (gg-mm-aaaa): ")
    datap = datetime.strptime(giornop, "%d-%m-%Y")

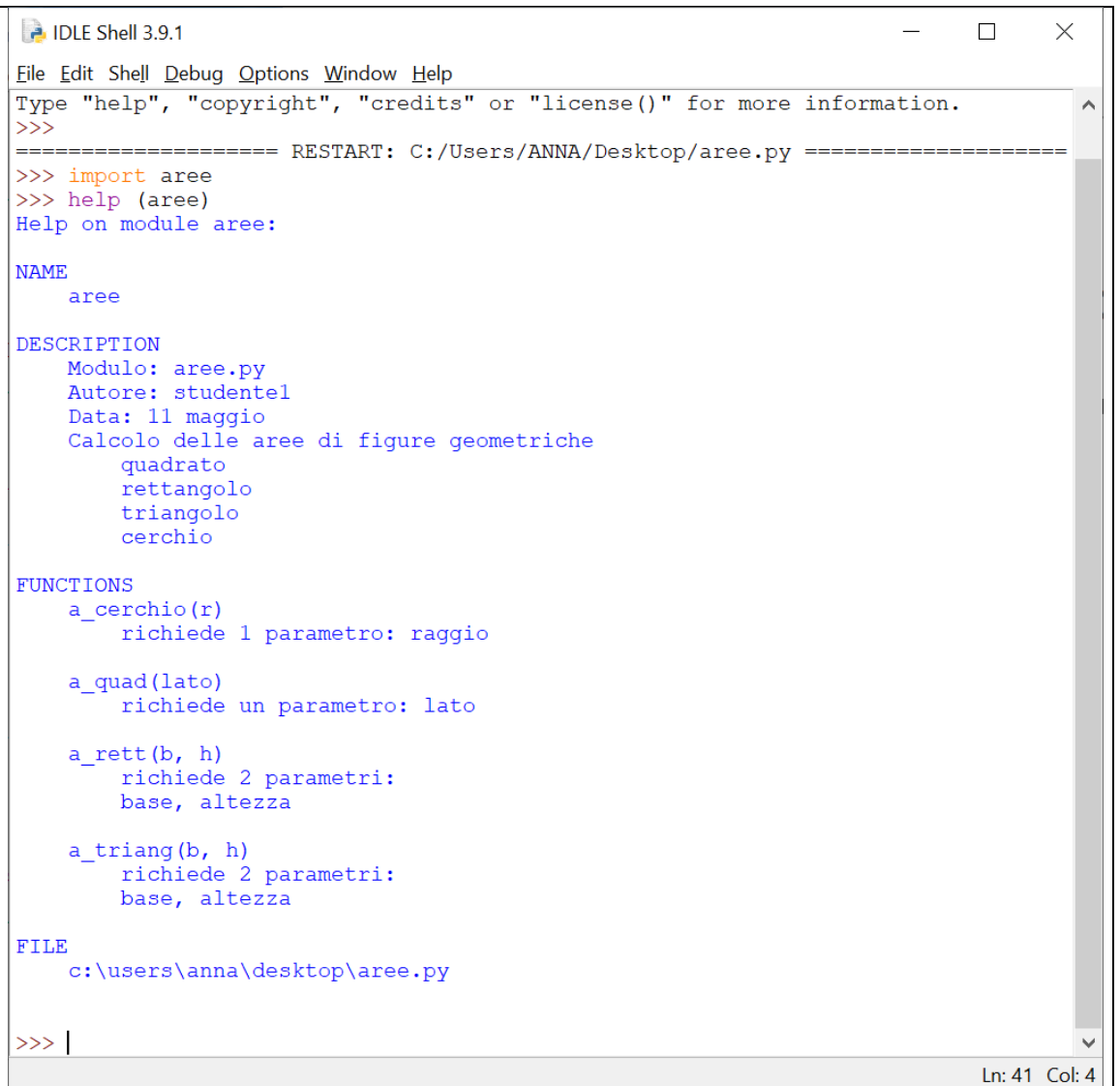
    # calcolo data di scadenza a 30 giorni
    datas = datap + timedelta(days = 30)

    # stampa
    print("Da consumarsi preferibilmente entro:")
    print(datas.strftime("%d %B %Y"))

# istruzioni di avvio del programma
main()
```

```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
== RESTART: C:/Users/ANNA/Desktop/ex Python/data di scadenza di un alimento.py ==
Data di produzione (gg-mm-aaaa): 12-05-2019
Da consumarsi preferibilmente entro:
11 giugno 2019
>>> |
```


<p>La documentazione dei moduli</p>	<p>Le triple virgolette (oppure triplici apici) servono a delimitare le stringhe in modo che siano visualizzate su più righe, senza indicazione del carattere di continuazione (barra rovesciata).</p> <p>La stessa notazione è usata per definire le <i>docstring</i>.</p> <p>Una docstring è una stringa racchiusa tra triple virgolette (o tripli apici) che contiene brevi testi di documentazione dei moduli e funzioni.</p> <p>La funzione help() usa le <i>docstring</i> di un modulo e di ciascuna funzione in esso contenuta per fornire informazioni sul codice, come un manuale in linea per l'utente.</p>
<p>Coding</p>	<p>Definire e documentare un modulo con le funzioni per il calcolo delle aree di alcune figure geometriche.</p>  <pre> *aree.py - C:/Users/ANNA/Desktop/aree.py (3.9.1)* File Edit Format Run Options Window Help """ Modulo: aree.py Autore: studentel Data: 11 maggio Calcolo delle aree di figure geometriche quadrato rettangolo triangolo cerchio """ # funzione per l'area del quadrato def a_quad(lato): """richiede un parametro: lato """ area = lato * lato return area # funzione per l'area del rettangolo def a_rett(b, h): """richiede 2 parametri: base, altezza """ area = b * h return area # funzione per l'area del triangolo def a_triang(b, h): """richiede 2 parametri: base, altezza """ area = (b * h)/2 return area # funzione per l'area del cerchio def a_cerchio(r): """ richiede 1 parametro: raggio """ import math area = r * r * math.pi return area </pre> <p>Ln: 11 Col: 0</p> <p>Con i due seguenti comandi, l'interprete Python legge le <i>docstring</i> e fornisce all'utente le informazioni sul modulo e le funzioni in esso contenute.</p> <pre> >>> import aree >>> help(aree) </pre>



```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/ANNA/Desktop/aree.py =====
>>> import aree
>>> help (aree)
Help on module aree:

NAME
    aree

DESCRIPTION
    Modulo: aree.py
    Autore: studentel
    Data: 11 maggio
    Calcolo delle aree di figure geometriche
        quadrato
        rettangolo
        triangolo
        cerchio

FUNCTIONS
    a_cerchio(r)
        richiede 1 parametro: raggio

    a_quad(lato)
        richiede un parametro: lato

    a_rett(b, h)
        richiede 2 parametri:
        base, altezza

    a_triang(b, h)
        richiede 2 parametri:
        base, altezza

FILE
    c:\users\anna\desktop\aree.py

>>> |
```

n.b.:

Per produrre manuali in linea si possono usare i programmi di generazione automatica della documentazione, che ricavano le informazioni dalle *docstring* producendo in automatico pagine formattate, tipicamente in HTML. Esempi di programmi di questo tipo sono *Sphinx*, *Pydoc* e *Doxygen*.

Python ha 28 parole riservate

Python ha 28 parole riservate:

```
and      continue  else      for       import    not       raise
assert   def          except    from      in        or        return
break    del          exec      global    is        pass     try
class    elif        finally   if        lambda    print    while
```

Sarebbe meglio tenere questa lista a portata di mano: se l'interprete ha problemi con il nome che vuoi assegnare ad una variabile e non ne capisci il motivo, prova a controllare se si trova in questa lista.

