



Corso di Apprendistato Python

Mod. Python
Docente:
Tonino Petrulli



Obiettivi del corso

Acquisire competenze di base sul linguaggio di programmazione Python

Tonino Petrulli

Contenuti del corso

- Introduzione al linguaggio
- tipi di base
- assegnamento
- operatori aritmetici
- input
- funzioni di conversione da stringa a tipo numerico
- operatori booleani
- funzioni
- funzioni delle stringhe
- istruzioni di selezione (if-else-elif)
- istruzioni di iterazione
- generazione di numeri random
- Le sequenze
- operazioni di selezione sulle sequenze
- le liste e i dizionari
- lettura/scrittura su file

Tonino Petrulli

Come funziona un programma per computer?

Un programma rende utilizzabile un computer. Senza un programma, un computer, anche il più potente, non è altro che un oggetto. Allo stesso modo, senza suonatore, un pianoforte non è altro che una scatola di legno.

Un computer può eseguire solo operazioni estremamente semplici. Ad esempio, un computer non è in grado di comprendere da solo il valore di una complicata funzione matematica, anche se ciò non è al di là delle possibilità nel prossimo futuro.

I computer moderni possono solo valutare i risultati di operazioni fondamentali, come l'addizione o la divisione, ma possono farlo molto velocemente e ripetere queste azioni praticamente un numero qualsiasi di volte.

Immagina di voler conoscere la velocità media che hai raggiunto durante un lungo viaggio. Conosci la distanza, conosci il tempo, hai bisogno della velocità.

Naturalmente il computer sarà in grado di calcolarlo, ma non è consapevole di cose come la distanza, la velocità o il tempo. Pertanto è necessario istruire il computer a:

- accettare un numero che rappresenta la distanza;
- accettare un numero che rappresenta il tempo di viaggio;
- dividere il primo valore per il secondo e memorizzare il risultato in memoria;
- visualizzare il risultato (che rappresenta la velocità media) in un formato leggibile.

Queste quattro semplici azioni formano un programma. Naturalmente questi esempi non sono formalizzati e sono molto lontani da ciò che il computer può capire, ma sono sufficientemente buoni per essere tradotti in un linguaggio che il computer può accettare.

Il linguaggio è la parola chiave!

Linguaggi naturali e linguaggi di programmazione

Un linguaggio è un mezzo (e uno strumento) per esprimere e registrare pensieri

- Linguaggio del corpo
- Lingua madre

Computer:

- Linguaggio macchina

Un computer, anche il più sofisticato tecnicamente, è privo di qualsiasi traccia di intelligenza, risponde solo a una serie predeterminata di comandi noti.

Un insieme completo di comandi noti è chiamato elenco di istruzioni.

Così come le persone usano una serie di lingue molto diverse tra loro, anche le macchine hanno molti linguaggi diversi

Linguaggi

Si può dire che ogni linguaggio (automatico o naturale, non importa) è costituito dai seguenti elementi:

- **un alfabeto:** un insieme di simboli utilizzati per costruire le parole di una certa lingua (ad esempio, l'alfabeto latino per l'inglese, l'alfabeto cirillico per il russo, i kanji per il giapponese, e così via)
- **un lessico:** (anche detto dizionario) un insieme di parole che la lingua offre ai suoi utenti (ad esempio, la parola “computer” è presente nel dizionario della lingua inglese, mentre “cmoptrue” non lo è; la parola “chat” è presente sia nel dizionario inglese che in quello francese, ma i loro significati sono diversi)
- **sintassi:** insieme di regole (formali o informali, scritte o percepite intuitivamente) utilizzate per determinare se una certa stringa di parole forma una frase valida
- **semantica:** un insieme di regole che determinano se una certa frase ha senso (ad esempio, “Ho mangiato una ciambella” ha senso, mentre “Una ciambella mi ha mangiato” non lo ha).

Il linguaggio macchina è l'insieme più semplice e primario di simboli che possiamo utilizzare per impartire comandi a un computer. È la lingua madre del computer.

Linguaggi ad alto livello

Purtroppo, il linguaggio macchina è molto lontana dalla lingua madre umana

Abbiamo bisogno di un linguaggio in cui gli esseri umani possano scrivere i loro programmi e di un linguaggio che i computer possano usare per eseguire i programmi, un linguaggio che sia molto più complesso del linguaggio macchina e molto più semplice del linguaggio naturale.

Tali linguaggi sono spesso chiamati **linguaggi di programmazione di alto livello**. Sono almeno in parte simili a quelli naturali, in quanto utilizzano simboli, parole e convenzioni leggibili dall'uomo.

Un programma scritto in un linguaggio di programmazione di alto livello è chiamato **codice sorgente** (in contrasto con il codice macchina eseguito dai computer). Allo stesso modo, il file contenente il codice sorgente è chiamato **file sorgente**.

Compilazione vs. interpretazione

Esistono due modi diversi per trasformare un programma da un linguaggio di programmazione di alto livello in linguaggio macchina:

COMPILAZIONE - il programma sorgente viene tradotto una volta (ma questo atto deve essere ripetuto ogni volta che si modifica il codice sorgente) ottenendo un file (ad esempio, un file .exe se il codice è destinato a essere eseguito in MS Windows) contenente il codice macchina; ora è possibile distribuire il file in tutto il mondo; il programma che esegue questa traduzione è chiamato compilatore o traduttore;

INTERPRETAZIONE - l'utente (o qualsiasi altro utente del codice) può tradurre il programma sorgente ogni volta che deve essere eseguito; il programma che esegue questo tipo di trasformazione è chiamato interprete, in quanto interpreta il codice ogni volta che deve essere eseguito; ciò significa anche che non si può semplicemente distribuire il codice sorgente così com'è, perché l'utente finale ha bisogno anche dell'interprete per eseguirlo.

Cosa fa l'interprete?

leggi-verifica-esegui

L'interprete

- legge il codice sorgente nel modo comune nella cultura occidentale: dall'alto in basso e da sinistra a destra.
- Controlla che ogni riga sia corretta (utilizzando i quattro aspetti trattati in precedenza).

Se l'interprete trova un errore, termina immediatamente il suo lavoro. L'unico risultato in questo caso è un messaggio di errore.

L'interprete informa l'utente su dove si trova l'errore e su cosa lo ha causato. Tuttavia, questi messaggi possono essere fuorvianti, poiché l'interprete non è in grado di seguire le vostre esatte intenzioni e può rilevare gli errori a una certa distanza dalle loro cause reali.

Se la riga sembra buona, l'interprete tenta di eseguirla

Compilazione vs interpretazione

	COMPILAZIONE	INTERPRETAZIONE
Vantaggi	<p>l'esecuzione del codice tradotto è solitamente più veloce;</p> <p>solo l'utente deve avere il compilatore - l'utente finale può usare il codice anche senza;</p> <p>il codice tradotto è memorizzato in linguaggio macchina - poiché è molto difficile da capire, è probabile che le vostre invenzioni e i vostri trucchi di programmazione rimangano segreti.</p>	<p>è possibile eseguire il codice non appena lo si è completato - non ci sono ulteriori fasi di traduzione;</p> <p>il codice viene memorizzato utilizzando il linguaggio di programmazione, non quello della macchina - questo significa che può essere eseguito su computer che utilizzano linguaggi macchina diversi; non è necessario compilare il codice separatamente per ogni diversa architettura.</p>

Compilazione vs interpretazione

	COMPILAZIONE	INTERPRETAZIONE
Svantaggi	<p>la compilazione in sé può essere un processo che richiede molto tempo - potreste non essere in grado di eseguire il vostro codice immediatamente dopo qualsiasi modifica; è necessario disporre di tanti compilatori quante sono le piattaforme hardware su cui si vuole che il codice venga eseguito.</p>	<p>non aspettatevi che l'interpretazione porti il vostro codice ad alta velocità: il vostro codice condividerà la potenza del computer con l'interprete, quindi non potrà essere veramente veloce; sia voi che l'utente finale dovete avere l'interprete per eseguire il vostro codice.</p>

Che cos'è Python?

Python è un linguaggio di programmazione di alto livello, interpretato, orientato agli oggetti, ampiamente utilizzato per la programmazione generale.

Sebbene il pitone sia conosciuto come un grosso serpente, il nome del linguaggio di programmazione Python deriva da una vecchia serie di sketch televisivi della BBC intitolata Monty Python's Flying Circus.

All'apice del successo, i Monty Python eseguivano i loro sketch al pubblico dal vivo in tutto il mondo.

Chi ha creato Python?

Una delle caratteristiche più sorprendenti di Python è che si tratta del lavoro di una sola persona. Di solito, i nuovi linguaggi di programmazione vengono sviluppati e pubblicati da grandi aziende che impiegano molti professionisti e, a causa delle regole sul copyright, è molto difficile fare i nomi delle persone coinvolte nel progetto. Python è un'eccezione.

Non sono molti i linguaggi i cui autori sono noti per nome. Python è stato creato da Guido van Rossum, nato nel 1956 ad Haarlem, nei Paesi Bassi. Naturalmente, Guido van Rossum non ha sviluppato ed evoluto da solo tutti i componenti di Python.

La velocità con cui Python si è diffuso in tutto il mondo è il risultato del lavoro continuo di migliaia di programmatori (molto spesso anonimi), tester, utenti (molti dei quali non sono informatici) e appassionati, ma va detto che la primissima idea (il seme da cui Python è germogliato) è venuta a una sola testa, quella di Guido.

.

Un progetto di programmazione per hobby

Guido van Rossum:

‘Nel dicembre 1989, stavo cercando un progetto di programmazione “hobbistico” che mi tenesse occupato durante la settimana di Natale. Il mio ufficio (...) sarebbe stato chiuso, ma avevo un computer a casa e non molto altro a disposizione. Decisi di scrivere un interprete per il nuovo linguaggio di scripting a cui avevo pensato negli ultimi tempi: un discendente dell'ABC che sarebbe piaciuto agli hacker Unix/C. Ho scelto Python come titolo provvisorio per il progetto, essendo di umore leggermente irriverente (e un grande fan del Monty Python's Flying Circus).’

.

Obiettivi di Python

Nel 1999, Guido van Rossum ha definito i suoi obiettivi per Python:

- un linguaggio facile e intuitivo, potente quanto quelli dei principali concorrenti; open source, in modo che chiunque possa contribuire al suo sviluppo;
- un codice comprensibile come l'inglese; adatto alle attività quotidiane, consentendo tempi di sviluppo brevi.

Circa 20 anni dopo, è chiaro che tutte queste intenzioni sono state realizzate. Alcune fonti affermano che Python è il linguaggio di programmazione più diffuso al mondo, mentre altre sostengono che sia il secondo o il terzo.

In ogni caso, occupa ancora un posto di rilievo nella top ten del PYPL Popularity of Programming Language e del TIOBE Programming Community Index.

.

Cosa rende Python speciale?

- è facile da imparare - il tempo necessario per imparare Python è più breve rispetto a molti altri linguaggi; questo significa che è possibile iniziare la programmazione vera e propria più velocemente;
- è facile da insegnare - il carico di lavoro didattico è minore rispetto a quello richiesto da altri linguaggi; ciò significa che l'insegnante può porre maggiore enfasi sulle tecniche generali di programmazione (indipendenti dal linguaggio), senza sprecare energie in trucchi esotici, strane eccezioni e regole incomprensibili;
- è facile da usare per scrivere nuovo software - spesso è possibile scrivere codice più velocemente usando Python;
- è facile da capire - spesso è anche più facile capire più velocemente il codice di qualcun altro se è scritto in Python;
- è facile da ottenere, installare e distribuire - Python è gratuito, aperto e multiplatforma, cosa che non tutti i linguaggi possono vantare.

Naturalmente, Python ha anche i suoi svantaggi:

non è un demone della velocità - Python non offre prestazioni eccezionali;

in alcuni casi può essere resistente ad alcune tecniche di test più semplici - questo può significare che il debug del codice Python può essere più difficile che con altri linguaggi.

Dove possiamo vedere Python in azione?

Lo vediamo ogni giorno e quasi ovunque. Viene utilizzato ampiamente per implementare servizi Internet complessi come i motori di ricerca, gli strumenti di archiviazione cloud, i social media e così via. Ogni volta che si utilizza uno di questi servizi, si è in realtà molto vicini a Python, anche se non lo si direbbe.

Molti strumenti di sviluppo sono implementati in Python. Sempre più applicazioni di uso quotidiano vengono scritte in Python. Molti scienziati hanno abbandonato i costosi strumenti proprietari per passare a Python. Molti tester di progetti IT hanno iniziato a usare Python per eseguire procedure di test ripetibili. L'elenco è lungo.

Perché non Python?

Nonostante la crescente popolarità di Python, ci sono ancora alcune nicchie in cui Python è assente, o è visto raramente:

- programmazione a basso livello (a volte chiamata programmazione “close to metal”): se si vuole implementare un driver o un motore grafico estremamente efficace, non si userebbe Python;
- applicazioni per dispositivi mobili: anche se questo territorio è ancora in attesa di essere conquistato da Python, è molto probabile che un giorno lo sarà.

Python 2 vs. Python 3

Esistono due tipi principali di Python, denominati Python 2 e Python 3.

Python 2 è una versione più vecchia del Python originale. Il suo sviluppo è stato intenzionalmente interrotto, anche se questo non significa che non ci siano aggiornamenti. Al contrario, gli aggiornamenti vengono rilasciati regolarmente, ma non sono destinati a modificare il linguaggio in modo significativo. Piuttosto correggono eventuali bug e falle di sicurezza appena scoperte. Il percorso di sviluppo di Python 2 è già arrivato a un punto morto, ma Python 2 stesso è ancora molto vivo.

Python 3 è la versione più recente (per essere precisi, quella attuale) del linguaggio. Sta attraversando il proprio percorso evolutivo, creando i propri standard e le proprie abitudini.

Queste due versioni di Python **non sono compatibili tra loro**. Gli script Python 2 non funzionano in un ambiente Python 3 e viceversa, quindi se si vuole che il vecchio codice Python 2 venga eseguito da un interprete Python 3, l'unica soluzione possibile è quella di riscriverlo, ovviamente non da zero, in quanto gran parte del codice può rimanere intatto, ma bisogna rivedere tutto il codice per trovare tutte le possibili incompatibilità. Purtroppo questo processo non può essere completamente automatizzato.

È troppo difficile, troppo lungo, troppo costoso e troppo rischioso migrare una vecchia applicazione Python 2 su una nuova piattaforma. È possibile che la riscrittura del codice introduca nuovi bug. È più facile e sensato lasciare in pace questi sistemi e migliorare l'interprete esistente, invece di cercare di lavorare all'interno del codice sorgente già funzionante.

Python alias CPython

Oltre a Python 2 e Python 3, esiste più di una versione di ciascuno di essi.

Prima di tutto, ci sono i Python che sono mantenuti dalle persone riunite intorno alla PSF (Python Software Foundation), una comunità che ha lo scopo di sviluppare, migliorare, espandere e divulgare Python e il suo ambiente. Il presidente della PSF è Guido van Rossum in persona e per questo motivo questi Python sono chiamati canonici. Sono anche considerati dei Python di riferimento, in quanto qualsiasi altra implementazione del linguaggio deve seguire tutti gli standard stabiliti dal PSF.

Guido van Rossum utilizzò il linguaggio di programmazione “C” per implementare la primissima versione del suo linguaggio e questa decisione è ancora in vigore. Tutti i Python provenienti dalla PSF sono scritti in linguaggio “C”. Le ragioni di questo approccio sono molteplici e hanno molte conseguenze. Una di queste (probabilmente la più importante) è che grazie ad essa Python può essere facilmente portato e migrato su tutte le piattaforme che hanno la possibilità di compilare ed eseguire programmi in linguaggio “C” (praticamente tutte le piattaforme hanno questa caratteristica, il che apre molte opportunità di espansione per Python).

Per questo motivo l'implementazione di PSF viene spesso chiamata CPython. È il Python più influente tra tutti i Python del mondo.

Cython

Un altro membro della famiglia Python è Cython.

Cython è una delle possibili soluzioni alla caratteristica più dolorosa di Python: la mancanza di efficienza. Calcoli matematici grandi e complessi possono essere facilmente codificati in Python (molto più facilmente che in “C” o in qualsiasi altro linguaggio tradizionale), ma l'esecuzione del codice risultante può richiedere molto tempo.

Come si conciliano queste due contraddizioni? Una soluzione è scrivere le proprie idee matematiche in Python e, quando si è assolutamente sicuri che il codice sia corretto e produca risultati validi, tradurlo in “C”. Certamente, il “C” funzionerà molto più velocemente del Python puro.

Questo è lo scopo di Cython: tradurre automaticamente il codice Python (pulito e chiaro, ma non troppo veloce) in codice “C” (complicato e loquace, ma agile).

Jython

Un'altra versione di Python si chiama Jython.



“J” sta per ‘Java’. Immaginate un Python scritto in Java invece che in C. Questo è utile, ad esempio, se sviluppate sistemi grandi e complessi scritti interamente in Java e volete aggiungervi un po' di flessibilità Python. Il tradizionale CPython potrebbe essere difficile da integrare in un ambiente di questo tipo, poiché C e Java vivono in mondi completamente diversi e non condividono molte idee comuni.

Jython può comunicare con l'infrastruttura Java esistente in modo più efficace. Per questo motivo alcuni progetti lo trovano utile e necessario.

Nota: l'attuale implementazione di Jython segue gli standard di Python 2. Non esiste un'implementazione Jython conforme. Per ora non esiste un Jython conforme a Python 3.



PyPy, un Python all'interno di un Python. In altre parole, rappresenta un ambiente Python scritto in un linguaggio simile a Python, chiamato RPython (Restricted Python). Si tratta in realtà di un sottoinsieme di Python.

Il codice sorgente di PyPy non viene eseguito in modo interpretativo, ma viene tradotto nel linguaggio di programmazione C e poi eseguito separatamente.

Questo è utile perché se si vuole testare una nuova funzionalità che potrebbe essere introdotta (ma non necessariamente) nell'implementazione Python tradizionale, è più facile verificarla con PyPy che con CPython. Questo è il motivo per cui PyPy è piuttosto uno strumento per chi sviluppa Python che per il resto degli utenti. Questo non rende PyPy meno importante o meno serio di CPython, ovviamente.

Inoltre, PyPy è compatibile con il linguaggio Python 3.

Come ottenere Python e come utilizzarlo

Esistono diversi modi per ottenere una copia di Python 3, a seconda del sistema operativo utilizzato.

Gli utenti Linux molto probabilmente hanno Python già installato: questo è lo scenario più probabile, poiché l'infrastruttura di Python è utilizzata intensamente da molti componenti del sistema operativo Linux.

Se siete utenti Linux, aprite il terminale/console e digitate:

```
python3
```

al prompt della shell, premere Invio e attendere.

Se viene visualizzato qualcosa di simile a questo:

```
Python 3.4.5 (default, Jan 12 2017, 02:28:40)
[GCC 4.2.1 Compatible Clang 3.7.1 (tags/RELEASE_371/final)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

allora non dovrete fare nient'altro.

Tutti gli utenti non Linux possono scaricare una copia da <https://www.python.org/downloads/>.

Scaricare e installare Python

Poiché il browser indica al sito il sistema operativo utilizzato, l'unico passo da compiere è quello di cliccare sulla versione di Python desiderata.


In questo caso, selezionare Python 3. Il sito offre sempre la versione più recente.

Se siete utenti di Windows, avviate il file .exe scaricato e seguite tutti i passaggi.

Per ora lasciate le impostazioni predefinite suggerite dal programma di installazione, con un'eccezione: guardate la casella di controllo Add Python 3.x to PATH e selezionatela.

Questo renderà le cose più semplici.

Se siete utenti di macOS, è possibile che una versione di Python 2 sia già preinstallata sul vostro computer, ma poiché lavoreremo con Python 3, dovrete comunque scaricare e installare il relativo file .pkg dal sito di Python.



The screenshot shows the Python.org homepage. At the top, there's a navigation bar with links for Python, PEP, Docs, PyPI, Jobs, and Community. Below this is a search bar and a 'Socialize' button. The main content area features a large banner with the text 'Download the latest version for Windows' and a button 'Download Python 3.7.0'. To the right of the banner is an illustration of two parachutes carrying boxes. Below the banner, there's a section for 'Looking for a specific release?' with a table of Python releases by version number.

Release version	Release date	Download	Click for more
Python 3.5.6	2018-08-02	Download	Release Notes
Python 3.4.9	2018-08-02	Download	Release Notes
Python 3.7.0	2018-06-27	Download	Release Notes

Iniziare a lavorare con Python

Per iniziare il vostro lavoro, avete bisogno dei seguenti strumenti:

un **editor** che vi supporti nella scrittura del codice (dovrebbe avere alcune caratteristiche speciali, non disponibili in strumenti semplici); questo editor dedicato vi darà qualcosa in più rispetto alla dotazione standard del sistema operativo;

una **console** in cui lanciare il codice appena scritto e fermarlo forzatamente quando va fuori controllo;

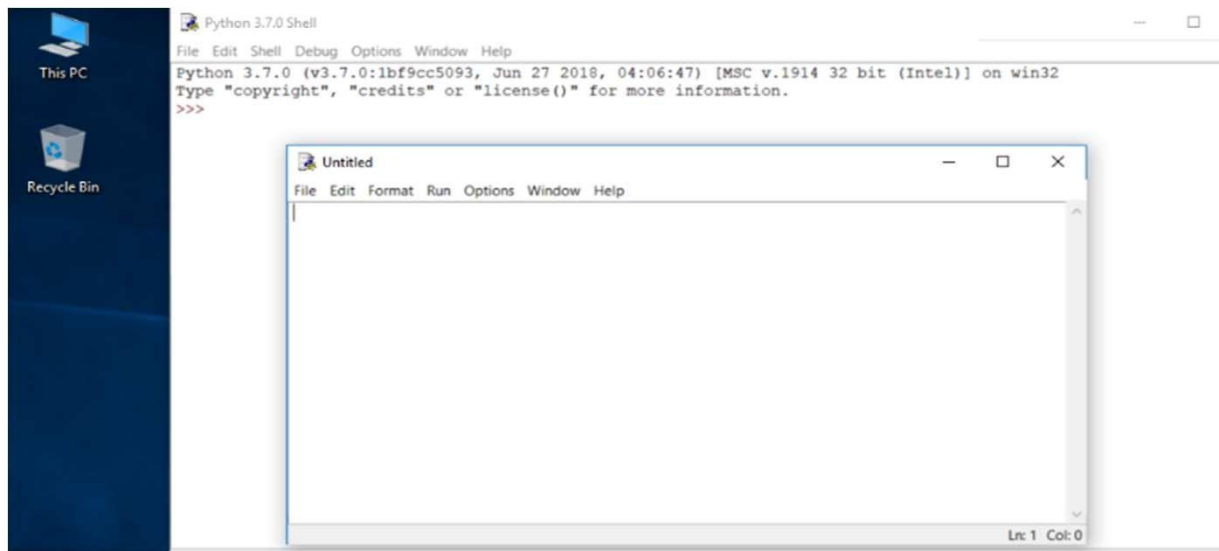
uno strumento chiamato debugger, in grado di lanciare il codice passo dopo passo e di consentirne l'ispezione in ogni momento dell'esecuzione.

Oltre ai suoi numerosi componenti utili, l'installazione standard di Python 3 contiene un'applicazione molto semplice ma estremamente utile, chiamata IDLE.

IDLE è un acronimo: **Integrated Development and Learning Environment** (ambiente di sviluppo e apprendimento integrato).

Iniziare a lavorare con Python

Il primo passo è creare un nuovo file sorgente e riempirlo di codice. Fate clic su File nel menu di IDLE e scegliete Nuovo file.



Iniziare a lavorare con Python

Come si può vedere, IDLE apre una nuova finestra. È possibile utilizzarla per scrivere e modificare il codice. Questa è la finestra dell'editor. Il suo unico scopo è quello di essere un luogo di lavoro in cui viene trattato il codice sorgente. Non confondete la finestra dell'editor con la finestra della shell. Esse svolgono funzioni diverse. La finestra dell'editor è attualmente senza titolo, ma è buona norma iniziare il lavoro dando un nome al file sorgente.

Fate clic su File (nella nuova finestra), quindi su Salva con nome..., selezionate una cartella per il nuovo file (il desktop è un buon posto per i primi tentativi di programmazione) e scegliete un nome per il nuovo file.

Nota: non impostare alcuna estensione per il nome del file che si intende utilizzare. Python ha bisogno che i suoi file abbiano l'estensione .py, quindi bisogna affidarsi alle impostazioni predefinite della finestra di dialogo. L'uso dell'estensione standard .py consente al sistema operativo di aprire correttamente questi file.

Come scrivere ed eseguire il vostro primo programma

Ora inserite una sola riga nella finestra dell'editor appena aperta e denominata.

La riga assomiglia a questa:

```
print("Hello World! ")
```

Osservate attentamente le virgolette. Si tratta della forma più semplice di virgolette (neutre, diritte, mute, ecc.) comunemente usata nei file sorgente. Non cercate di usare le virgolette tipografiche (curve, ricce, intelligenti, ecc.), usate dai processori di testo avanzati, perché **Python non le accetta**.

Salvare il file (File -> Save) ed eseguire il programma (Run -> Run Module).

Se tutto va bene e non ci sono errori nel codice, la finestra della console mostrerà gli effetti causati dall'esecuzione del programma.

Come scrivere ed eseguire il vostro primo programma

Chiudere ora entrambe le finestre e tornare al desktop.
Avviare nuovamente IDLE.

Fate clic su File, Apri, puntate al file salvato in precedenza e lasciate che IDLE lo legga.
Provate a eseguirlo di nuovo premendo F5 quando la finestra dell'editor è attiva.
Come si può notare, IDLE è in grado di salvare il codice e di recuperarlo quando se ne ha bisogno.

IDLE contiene un'ulteriore e utile funzione.

Innanzitutto, rimuovete la parentesi di chiusura.
Quindi inserire nuovamente la parentesi.

Ogni volta che si inserisce una parentesi di chiusura nel programma, IDLE mostra la parte di testo limitata da una coppia di parentesi corrispondenti. Questo aiuta a ricordare di inserirle in coppia.
Rimuovete nuovamente la parentesi di chiusura. Il codice diventa errato. Ora contiene un errore di sintassi. IDLE non dovrebbe permetterne l'esecuzione.

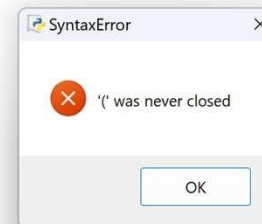
Riprovate a eseguire il programma. IDLE vi ricorderà di salvare il file modificato

Come scrivere ed eseguire il vostro primo programma

Il messaggio di errore ci informa che c'è una parentesi che non è stata chiusa

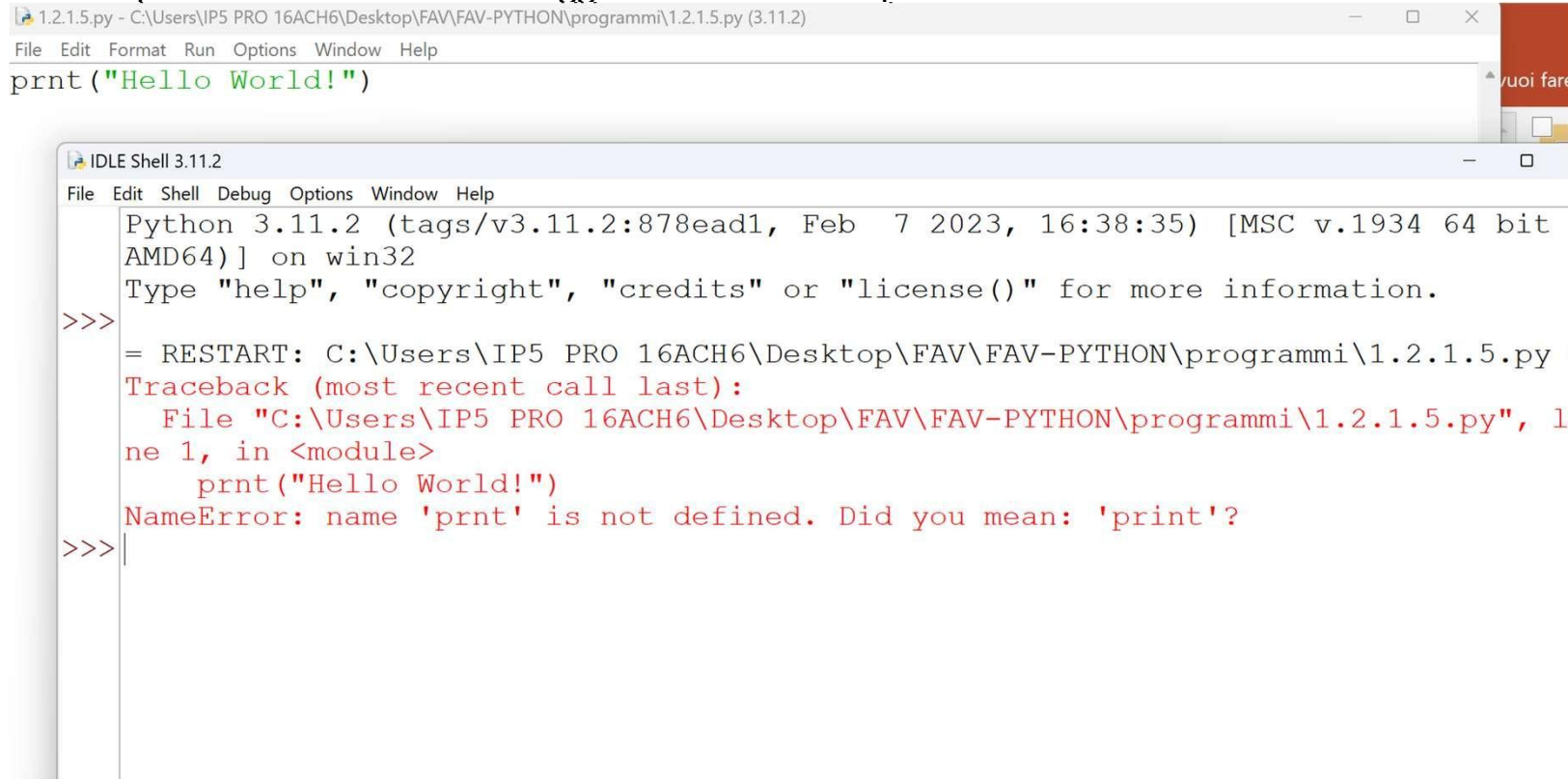


The screenshot shows a Python IDE window titled "1.2.1.5.py - C:\Users\IP5 PRO 16ACH6\Desktop\FAV\FAV-PYTHON\programmi\1.2.1.5.py (3.11.2)". The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code editor contains the line `print("Hello World!"`. A red squiggly line under the opening parenthesis indicates a syntax error.



Come scrivere ed eseguire il vostro primo programma

Rimettete ora la parentesi chiusa e rimuovete un carattere dalla parola print. Il messaggio di errore ci avvisa che Il nome prnt non è definito e ci suggerisce anche una possibile correzione.



The image shows a screenshot of a Python IDE window titled "1.2.1.5.py - C:\Users\IP5 PRO 16ACH6\Desktop\FAV\FAV-PYTHON\programmi\1.2.1.5.py (3.11.2)". The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code in the editor is `prnt("Hello World!")`. Below the editor is an "IDLE Shell 3.11.2" window. The shell displays the Python version and system information, followed by a prompt `>>>`. The user has entered `= RESTART: C:\Users\IP5 PRO 16ACH6\Desktop\FAV\FAV-PYTHON\programmi\1.2.1.5.py :`. The shell then shows a `Traceback (most recent call last):` error. The error details are: `File "C:\Users\IP5 PRO 16ACH6\Desktop\FAV\FAV-PYTHON\programmi\1.2.1.5.py", line 1, in <module>` followed by `prnt("Hello World!")`. The final error message is `NameError: name 'prnt' is not defined. Did you mean: 'print'?`. The prompt `>>>` is shown again at the bottom of the shell window.

```
1.2.1.5.py - C:\Users\IP5 PRO 16ACH6\Desktop\FAV\FAV-PYTHON\programmi\1.2.1.5.py (3.11.2)
File Edit Format Run Options Window Help
prnt("Hello World!")

IDLE Shell 3.11.2
File Edit Shell Debug Options Window Help
Python 3.11.2 (tags/v3.11.2:878ead1, Feb 7 2023, 16:38:35) [MSC v.1934 64 bit
AMD64] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\IP5 PRO 16ACH6\Desktop\FAV\FAV-PYTHON\programmi\1.2.1.5.py :
Traceback (most recent call last):
  File "C:\Users\IP5 PRO 16ACH6\Desktop\FAV\FAV-PYTHON\programmi\1.2.1.5.py", line 1, in <module>
    prnt("Hello World!")
NameError: name 'prnt' is not defined. Did you mean: 'print'?
>>>
```

Come scrivere ed eseguire il vostro primo programma

La finestra dell'editor non fornirà alcuna informazione utile sull'errore, ma le finestre della console potrebbero farlo.

Il messaggio (in rosso) si presenta (nelle righe successive):

- il traceback (che è il percorso che il codice compie attraverso le diverse parti del programma - si può ignorare per ora, perché è vuoto in un codice così semplice);
- la posizione dell'errore (il nome del file contenente l'errore, il numero di riga e il nome del modulo); nota: il numero può essere fuorviante, poiché Python di solito mostra il punto in cui nota per la prima volta gli effetti dell'errore, non necessariamente l'errore stesso;
- il contenuto della riga errata; nota: la finestra dell'editor di IDLE non mostra i numeri di riga, ma visualizza la posizione corrente del cursore nell'angolo in basso a destra; usatela per individuare la riga errata in un codice sorgente lungo;
- il nome dell'errore e una breve spiegazione.

Sperimentate la creazione di nuovi file e l'esecuzione del codice. Provate a trasmettere sullo schermo un messaggio diverso. Provate a rovinare e correggere il vostro codice e vedete cosa succede.

Tipi di dati

variabili

operazioni di base di input-output

operatori di base

Il vostro primo programma

Riaprite l'IDLE ed il programma con la riga di codice

```
print("Hello World! ")
```

Come potete vedere, il primo programma è composto dalle seguenti parti:

la parola print;

una parentesi di apertura;

una virgoletta;

una riga di testo: Hello World!

un'altra virgoletta;

una parentesi di chiusura.

Ognuno di questi elementi svolge un ruolo molto importante nel codice

La funzione print()

La parola print che si vede qui è il nome di una funzione

Una funzione (in questo contesto) è una parte separata del codice del computer in grado di:

- causare un effetto (ad esempio, inviare un testo al terminale, creare un file, disegnare un'immagine, riprodurre un suono, ecc;
- valutare un valore (ad esempio, la radice quadrata di un valore o la lunghezza di un testo) e restituirlo come risultato della funzione; questo è ciò che rende le funzioni Python parenti dei concetti matematici.

Inoltre, molte funzioni Python possono fare le due cose insieme.

Da dove vengono le funzioni?

Possono provenire da Python stesso; la funzione print è uno di questi casi; una funzione di questo tipo è un valore aggiunto ricevuto insieme a Python e al suo ambiente (è incorporata); non è necessario fare nulla di speciale (ad esempio, chiedere qualcosa a qualcuno) se si desidera utilizzarla;

possono provenire da uno o più componenti aggiuntivi di Python, chiamati moduli; alcuni moduli vengono forniti con Python, altri possono richiedere un'installazione separata; in ogni caso, tutti devono essere collegati esplicitamente al vostro codice

La funzione `print()`

Come abbiamo detto prima, una funzione può avere:

un effetto;

un risultato.

C'è anche un terzo componente della funzione, molto importante: gli argomenti.

Le funzioni matematiche di solito prendono un argomento, ad esempio $\sin(x)$ prende una x , che è la misura di un angolo.

Le funzioni di Python, invece, sono più versatili. A seconda delle esigenze individuali, possono accettare un numero qualsiasi di argomenti, quanti sono necessari per svolgere i loro compiti.

Nota: qualsiasi numero include anche lo zero; alcune funzioni Python non hanno bisogno di alcun argomento.

La funzione `print()`

In questo esempio, l'unico argomento fornito alla funzione `print()` è una stringa:

```
print("Hello World! ")
```

Come si può notare, la stringa è delimitata da virgolette; in effetti, le virgolette creano la stringa, tagliano una parte del codice e le assegnano un significato diverso.

Si può immaginare che le virgolette dicano qualcosa come: il testo tra noi non è codice. Non è destinato a essere eseguito e va preso così com'è.

Quasi tutto ciò che viene inserito all'interno delle virgolette verrà preso alla lettera, non come codice, ma come dati.

La funzione `print()`

Finora abbiamo imparato a conoscere due parti importanti del codice: la funzione e la stringa. Ne abbiamo parlato in termini di sintassi, ma ora è il momento di parlarne in termini di semantica.

Il nome della funzione (`print` in questo caso), insieme alle parentesi e agli argomenti, costituisce l'invocazione della funzione.

Che cosa succede quando Python incontra un'invocazione come questa qui sotto?

`nome_funzione(argomento)`

- controlla se il nome specificato è legale (sfoglia i suoi dati interni per trovare una funzione esistente con quel nome; se questa ricerca fallisce, Python interrompe il codice);
 - verifica se il numero di argomenti richiesti dalla funzione consente di invocare la funzione in questo modo (ad esempio, se una funzione specifica richiede esattamente due argomenti, qualsiasi invocazione che fornisca un solo argomento sarà considerata errata e interromperà l'esecuzione del codice);
 - abbandona per un attimo il codice e salta alla funzione che si vuole invocare; naturalmente, prende anche gli argomenti e li passa alla funzione;
 - la funzione esegue il suo codice, provoca l'effetto desiderato (se c'è), valuta il risultato desiderato (se c'è) e termina il suo compito;
- infine, Python ritorna al codice (al punto immediatamente successivo all'invocazione) e riprende la sua esecuzione.

LAB-01

Tempo stimato: 5-10 minuti

Livello di difficoltà: Molto facile

Obiettivi

familiarizzare con la funzione `print()` e le sue capacità di formattazione;
sperimentare il codice Python.

Scenario

Il comando `print()`, una delle direttive più semplici di Python, stampa semplicemente una riga sullo schermo.

Nel primo laboratorio:

utilizzate la funzione `print()` per stampare sullo schermo la riga `Ciao, Python!` Utilizzate i doppi apici intorno alla stringa;

Fatto questo, usate di nuovo la funzione `print()`, ma questa volta stampate il vostro nome;

rimuovete i doppi apici ed eseguite il codice. Osservate la reazione di Python. Che tipo di errore viene lanciato?

Poi, rimuovete le parentesi, rimettete i doppi apici ed eseguite nuovamente il codice. Che tipo di errore viene lanciato questa volta?

Sperimentate il più possibile. Cambiate le doppie virgolette in singole, usate più funzioni `print()` sulla stessa riga e poi su righe diverse. Vedete cosa succede.

La sintassi di Python

A differenza della maggior parte dei linguaggi di programmazione, Python richiede che non ci sia più di un'istruzione in una riga.

Una riga può essere vuota (cioè può non contenere alcuna istruzione), ma non può contenere due, tre o più istruzioni. Questo è severamente vietato.

Nota: Python fa un'eccezione a questa regola: permette che un'istruzione si distribuisca su più righe (il che può essere utile quando il codice contiene costruzioni complesse).

La funzione print

Analizziamo il codice seguente

Codice

```
print("Il ragnetto si arrampicò sulla tromba d'acqua.")  
print("La pioggia è scesa e ha spazzato via il ragno.")
```

Eseguitelo e notate cosa vedete nella console.

La console di Python dovrebbe ora avere questo aspetto:

Il ragnetto si arrampicò sulla tromba d'acqua.

La pioggia è scesa e ha spazzato via il ragno.

Questa è una buona occasione per fare alcune osservazioni:

- il programma invoca due volte la funzione `print()` e si possono vedere due righe separate nella console - questo significa che `print()` inizia il suo output da una **nuova riga** ogni volta che inizia la sua esecuzione; si può cambiare questo comportamento, ma si può anche usarlo a proprio vantaggio;
- ogni invocazione di `print()` contiene una stringa diversa come argomento e il contenuto della console lo riflette - questo significa che le **istruzioni del codice vengono eseguite nello stesso ordine in cui sono state inserite** nel file sorgente; nessuna istruzione successiva viene eseguita finché la precedente non è stata completata

La funzione print

Al codice precedente aggiungete una `print()` senza parametri

```
print("Il ragnetto si arrampicò sulla tromba d'acqua.")  
print()  
print("La pioggia è scesa e ha spazzato via il ragno.")
```

Eseguitelo e notate cosa vedete nella console.

Come si può vedere, l'invocazione vuota `print()` non è così vuota come ci si aspettava: produce una riga vuota, oppure (anche questa interpretazione è corretta) il suo output è solo un newline.

Questo non è l'unico modo per produrre un newline nella console di output.

La funzione print() - i caratteri di escape e newline

Modifichiamo nuovamente il codice.

```
print("The itsy bitsy spider\nclimbed up the waterspout.")  
print()  
print("Down came the rain\nand washed the spider out.")
```

Ci sono due cambiamenti molto sottili: abbiamo inserito una strana coppia di caratteri all'interno della rima. Hanno questo aspetto: \n.

È interessante notare che, mentre voi vedete due caratteri, Python ne vede uno solo.

Il backslash (\) ha un significato molto particolare quando viene usato all'interno delle stringhe: si chiama **carattere di escape**.

La funzione print() - i caratteri di escape e newline

La parola escape deve essere intesa in modo specifico: significa che la serie di caratteri nella stringa sfugge per un momento (un momento molto breve) per introdurre un'inclusione speciale.

In altre parole, il backslash non ha un significato in sé, ma è solo una sorta di annuncio che il carattere successivo al backslash ha un significato diverso.

La lettera n posta dopo il backslash deriva dalla parola **newline**.

Sia il backslash che la n formano un simbolo speciale chiamato carattere newline, che invita la console a iniziare una nuova riga di output.

Eseguite il codice. La console dovrebbe avere questo aspetto:

Il ragnetto
si arrampicò sulla tromba d'acqua.

Scese la pioggia
e lavò il ragno..

La funzione `print()` - i caratteri di escape e newline

Questa convenzione ha due importanti conseguenze:

1. Se si vuole inserire un solo backslash all'interno di una stringa, non bisogna dimenticare la sua natura di escape - bisogna raddoppiarlo, ad esempio, tale invocazione causerà un errore:

```
print("\")
```

mentre questa non lo farà:

```
print("\\")
```

2. Non tutte le coppie di escape (il backslash accoppiato a un altro carattere) hanno un significato.

Sperimentate il vostro codice nell'editor, eseguitelo e vedete cosa succede.

La funzione print() - utilizzo di più argomenti

Finora abbiamo testato il comportamento della funzione print() senza argomenti e con un argomento. Vale la pena di provare ad alimentare la funzione print() con più di un argomento.

Questo è ciò che stiamo per testare ora:

```
print("Il ragnetto", "si arrampicò", "la tromba d'acqua").
```

C'è una sola invocazione della funzione print(), ma contiene tre argomenti. Tutti sono stringhe. Gli argomenti sono separati da virgole. Eseguite il codice e vedete cosa succede.

La console dovrebbe ora mostrare il seguente testo:

Il ragnetto si è arrampicato sulla tromba d'acqua.

- Una funzione print() invocata con più di un argomento li fa uscire tutti su una riga;
- La funzione print() **inserisce di sua iniziativa** uno spazio tra gli argomenti inviati.

La funzione `print()`: il modo posizionale di passare gli argomenti

Il modo in cui passiamo gli argomenti alla funzione `print()` è il più comune in Python ed è chiamato modo posizionale (questo nome deriva dal fatto che il significato dell'argomento è dettato dalla sua posizione, ad esempio, il secondo argomento verrà inviato dopo il primo e non viceversa).

La funzione print() - gli argomenti delle parole chiave

Python offre un altro meccanismo per il passaggio di argomenti, che può essere utile quando si vuole convincere la funzione print() a cambiare un po' il suo comportamento.

Il meccanismo si chiama "argomenti per parole chiave". Il nome deriva dal fatto che il significato di questi argomenti non è dato dalla loro posizione, ma dalla parola speciale (parola chiave) usata per identificarli.

La funzione print() ha due argomenti parola chiave che possono essere utilizzati per i propri scopi. Il primo di essi si chiama end.

Nel codice seguente si può vedere un esempio molto semplice di utilizzo di un argomento parola chiave.

```
print("My name is", "Python.", end=" ")  
print("Monty Python.")
```


La funzione print() - gli argomenti delle parole chiave

Per utilizzarlo, è necessario conoscere alcune regole:

- un argomento di parola chiave è composto da tre elementi: una parola chiave che identifica l'argomento (qui end); un segno di uguale (=); un valore assegnato a quell'argomento;
- gli argomenti delle parole chiave devono essere messi dopo l'ultimo argomento posizionale (questo è molto importante).

Nel nostro esempio, abbiamo utilizzato l'argomento parola chiave end e lo abbiamo impostato su una stringa contenente uno spazio.

Eseguire il codice per vedere come funziona.

La console dovrebbe ora mostrare il seguente testo:

Il mio nome è Python. Monty Python.

Come si può vedere, l'argomento della parola chiave end determina i caratteri che la funzione print() invia all'output una volta raggiunta la fine dei suoi argomenti posizionali.

Il comportamento predefinito riflette la situazione in cui l'argomento della parola chiave end viene utilizzato implicitamente nel modo seguente: end="\n".

La funzione print() - gli argomenti delle parole chiave

Provate ora con il seguente codice:

```
print("My name is ", end="")  
print("Monty Python.")
```

Se guardate bene, vedrete che abbiamo usato l'argomento end, ma la stringa ad esso assegnata è vuota (non contiene alcun carattere).

Che cosa succederà ora? Eseguite il programma nell'editor per scoprirlo.

Poiché l'argomento end è stato impostato a zero, anche la funzione print() non produce nulla, una volta esauriti i suoi argomenti posizionali.

La console dovrebbe ora mostrare il seguente testo:

Il mio nome è Monty Python.

Nota: non sono state inviate linee nuove all'output.

La stringa assegnata all'argomento della parola chiave end può essere di qualsiasi lunghezza. Sperimentate!

La funzione print() - gli argomenti delle parole chiave

Abbiamo detto in precedenza che la funzione print() separa i suoi argomenti in uscita con degli spazi. Anche questo comportamento può essere modificato.

L'argomento della parola chiave che può fare questo si chiama sep (come separatore).

Scrivete il codice nell'editor ed eseguitelo.

```
print("My", "name", "is", "Monty", "Python.", sep="-")
```

L'argomento sep fornisce i seguenti risultati:

My-name-is-Monty-Python.

La funzione print() ora usa un trattino, invece di uno spazio, per separare gli argomenti in uscita.

Nota: il valore dell'argomento sep può essere anche una stringa vuota. Provate!

La funzione print() - gli argomenti delle parole chiave

Entrambi gli argomenti delle parole chiave possono essere mescolati in un'unica invocazione, proprio come qui nella finestra dell'editor.

L'esempio non ha molto senso, ma presenta visibilmente le interazioni tra end e sep.

```
print("My", "name", "is", sep="_", end="*")  
print("Monty", "Python.", sep="*", end="*\n")
```

Riuscite a prevedere l'output?

Eseguite il codice e vedete se corrisponde alle vostre previsioni.

LAB-02

Tempo stimato: 5-10 minuti

Livello di difficoltà: Molto facile

Obiettivi

familiarizzare con la funzione `print()` e le sue capacità di formattazione;
sperimentare il codice Python.

Scenario

Modificare la prima riga di codice nell'editor, utilizzando le parole chiave `sep` e `end`, in modo che corrisponda all'output previsto. Utilizzate le due funzioni `print()` nell'editor.

```
print("Programming", "Essentials", "in")  
print("Python")
```

Non modificate nulla nella seconda invocazione di `print()`.

Risultato previsto

Programmazione***Essenziali***in...Python

LAB-03

Tempo stimato: 5-15 minuti

Livello di difficoltà: Facile

Obiettivi

sperimentare con il codice Python esistente;

scoprire e correggere gli errori di sintassi di base;

familiarizzare con la funzione print() e le sue capacità di formattazione.

Scenario

Vi invitiamo a giocare con il codice che abbiamo scritto per voi e ad apportare alcune modifiche (forse anche distruttive). Sentitevi liberi di modificare qualsiasi parte del codice, ma a una condizione: imparate dai vostri errori e traete le vostre conclusioni.

```
print("  *")  
print(" * *")  
print(" *  *")  
print(" *   *")  
print("****  ****")  
print(" *   *")  
print(" *   *")  
print(" *****")
```

Continua...

LAB-03

Cercate di:

- ridurre al minimo il numero di invocazioni della funzione `print()` inserendo la sequenza `\n` nelle stringhe
- rendere la freccia due volte più grande (ma mantenendo le proporzioni)
- duplicare la freccia, affiancando le due frecce; nota: una stringa può essere moltiplicata utilizzando il seguente trucco: `"stringa" * 2` produrrà `"stringastringa"` (ve ne parleremo presto)
- rimuovete le virgolette e osservate attentamente la risposta di Python; prestate attenzione al punto in cui Python vede un errore: è questo il punto in cui l'errore esiste davvero?
- fare lo stesso con alcune parentesi;
- cambiate una qualsiasi delle parole `print` in qualcos'altro, differendo solo nel caso (ad esempio, `Print`) - cosa succede ora?
- sostituite alcune delle virgolette con degli apostrofi; osservate attentamente cosa succede.

Tipi di dato

Iniziamo con un semplice esperimento

```
print("2")
```

```
print(2)
```

Provate a eseguirlo.

Se tutto è andato bene, ora si dovrebbero vedere due linee identiche.

Che cosa significa?

In questo esempio, si incontrano due diversi tipi di letterali:

- una **stringa**, che già conoscete,
- e un numero **intero**, qualcosa di completamente nuovo.

La funzione `print()` li presenta esattamente nello stesso modo - questo esempio è ovvio, poiché anche la loro rappresentazione leggibile dall'uomo è la stessa. Internamente, nella memoria del computer, questi due valori sono memorizzati in modo completamente diverso: la stringa esiste solo come stringa, una serie di lettere.

Tipi di dato

Iniziamo con un semplice esperimento

```
print("2")
```

```
print(2)
```

Provate a eseguirlo.

Se tutto è andato bene, ora si dovrebbero vedere due linee identiche.

Che cosa significa?

In questo esempio, si incontrano due diversi tipi di letterali:

- una **stringa**, che già conoscete,
- e un numero **intero**, qualcosa di completamente nuovo.

La funzione `print()` li presenta esattamente nello stesso modo - questo esempio è ovvio, poiché anche la loro rappresentazione leggibile dall'uomo è la stessa. Internamente, nella memoria del computer, questi due valori sono memorizzati in modo completamente diverso: la stringa esiste solo come stringa, una serie di lettere.

Il numero viene convertito in una rappresentazione meccanica (un insieme di bit). La funzione `print()` è in grado di mostrarli entrambi in una forma leggibile per l'uomo.

Integer

I numeri gestiti dai computer sono di due tipi:

- **interi**, cioè quelli che sono privi della parte frazionaria;
- numeri **a virgola mobile** (o semplicemente **float**), che contengono (o sono in grado di contenere) la parte frazionaria.

La caratteristica del valore numerico che ne determina il tipo, l'intervallo e l'applicazione è chiamata **tipo**.

Se si codifica un letterale e lo si inserisce nel codice Python, la forma del letterale determina la rappresentazione (tipo) che Python utilizzerà per **memorizzarlo**.

Per ora, lasciamo da parte i numeri in virgola mobile e consideriamo la questione di come Python riconosce i numeri interi.

Prendiamo, ad esempio, il numero undici milioni e centoundicimila centoundici. Se prendeste in mano una matita in questo momento, scrivereste il numero in questo modo: 11.111.111, oppure così: 11.111.111, o anche così: 11 111 111.

È chiaro che questa disposizione facilita la lettura, soprattutto quando il numero è composto da molte cifre.

Tuttavia, Python non accetta cose come queste. È **vietato**. Ciò che Python permette, invece, è l'uso dei **trattini bassi** nei letterali numerici.*

Pertanto, è possibile scrivere questo numero sia in questo modo: 11111111, oppure così: 11_111_111.

NOTA *Python 3.6 ha introdotto i trattini bassi nei letterali numerici, consentendo di inserire singoli trattini bassi tra le cifre e dopo gli specificatori di base per migliorare la leggibilità. Questa funzione non è disponibile nelle versioni precedenti di Python.

Numeri interi: numeri ottali ed esadecimali

In Python esistono due convenzioni aggiuntive sconosciute al mondo della matematica. La prima ci permette di utilizzare i numeri in una rappresentazione **ottale**.

Se un numero intero è preceduto da un prefisso 0O o 0o (zero-o), viene trattato come un valore ottale. Ciò significa che il numero deve contenere solo cifre dell'intervallo [0..7].

0o123 è un numero **ottale** con un valore (decimale) pari a 83.

La funzione print() effettua la conversione automaticamente. Provate questo:

```
print(0o123)
```

La seconda convenzione consente di utilizzare numeri **esadecimali**. Tali numeri devono essere preceduti dal prefisso 0x o 0X (zero-x).

0x123 è un numero **esadecimale** con un valore (decimale) uguale a 291. La funzione print() può gestire anche questi valori. Provate questo:

```
print(0x123)
```

.

Floats

Ora è il momento di parlare di un altro tipo, progettato per rappresentare e memorizzare i numeri che (come direbbe un matematico) hanno una **frazione decimale non vuota**.

Sono i numeri che hanno (o possono avere) una parte frazionaria dopo la virgola e, sebbene tale definizione sia molto povera, è certamente sufficiente per ciò che vogliamo discutere.

Quando si usa un termine come *due e mezzo* o *meno zero virgola quattro*, si pensa a numeri che il computer considera **in virgola mobile**:

2.5

-0.4

Nota: anche se la vostra lingua madre preferisce usare una virgola al posto del punto nel numero, dovrete assicurarvi che il vostro **numero non contenga alcuna virgola**.

Python non lo accetterà o (in casi molto rari ma possibili) potrebbe fraintendere le vostre intenzioni, poiché la virgola stessa ha un suo significato riservato in Python.

Ma non dimenticate questa semplice regola: lo zero può essere omesso quando è l'unica cifra davanti o dopo la virgola.

In sostanza, è possibile scrivere il valore 0,4 come:

.4

Ad esempio, il valore di 4,0 potrebbe essere scritto come:

4.

Questo non cambierà né il suo tipo né il suo valore.

Floats

Quando si vogliono usare numeri molto grandi o molto piccoli, si può usare la **notazione scientifica**.

Prendiamo, ad esempio, la velocità della luce, espressa in *metri al secondo*. Scritta direttamente, sembrerebbe questa: 300000000.

Per evitare di scrivere tanti zeri, i libri di testo di fisica usano una forma abbreviata, che probabilmente avete già visto: 3×10^8 .

Si legge: tre volte dieci alla potenza di otto.

In Python, lo stesso effetto si ottiene in modo leggermente diverso: date un'occhiata:

3E8

La lettera E (si può usare anche la lettera e minuscola - deriva dalla parola **exponent**) è una rappresentazione concisa della frase *moltiplicata per dieci alla potenza di*.

Nota:

- l'**esponente** (il valore dopo la *E*) deve essere un numero intero;
- la **base** (il valore davanti alla *E*) può essere un numero intero.

Floats

Vediamo come questa convenzione viene utilizzata per registrare numeri molto piccoli (nel senso del loro valore assoluto, che è prossimo allo zero).

Una costante fisica chiamata *costante di Planck* (e indicata con h), secondo i libri di testo, ha il valore di: **6,62607 x 10⁻³⁴** .

Se si desidera utilizzarlo in un programma, è necessario scriverlo in questo modo:

6.62607E-34

Nota: il fatto che abbiate scelto una delle possibili forme di codifica dei valori float non significa che Python la presenterà nello stesso modo.

Python può talvolta scegliere una **notazione diversa** dalla vostra.

Ad esempio, supponiamo di aver deciso di utilizzare il seguente letterale float:

0.000000000000000000000001

Quando si esegue questo letterale in Python:

```
print(0.000000000000000000000001)
```

questo è il risultato:

1e-22

Python sceglie sempre **la forma più economica di presentazione del numero** e bisogna tenerne conto quando si creano i letterali.

Stringhe

Le stringhe vengono utilizzate quando è necessario elaborare del testo (come nomi di ogni tipo, indirizzi, romanzi, ecc.), non i numeri.

Ne sapete già qualcosa, ad esempio che le **stringhe hanno bisogno di virgolette** come i floats hanno bisogno di punti.

Questa è una stringa molto tipica: "Io sono una stringa".

Tuttavia, c'è un problema. Il problema è come codificare una citazione all'interno di una stringa già delimitata da virgolette.

Supponiamo di voler stampare un messaggio molto semplice che dice:

Mi piace "Monty Python"

Come farlo senza generare un errore? Ci sono due possibili soluzioni.

Il primo si basa sul concetto già noto di **carattere di escape**, che ricordiamo essere interpretato dal **backslash**. Il backslash può anche sfuggire alle virgolette. Una citazione preceduta da un backslash cambia il suo significato: non è un delimitatore, ma solo una citazione. Questo funzionerà come previsto:

```
print("Mi piace \"Monty Python\"")
```

Stringhe

Python può usare **un apostrofo al posto delle virgolette**. Entrambi i caratteri possono delimitare le stringhe, ma bisogna essere **coerenti**.

Se si apre una stringa con le doppie virgolette, bisogna chiuderla con doppie virgolette.

Se si inizia una stringa con un apostrofo, bisogna terminarla con un apostrofo.

Anche questo esempio può funzionare:

```
print('Mi piace "Monty Python"')
```

Nota: non è necessario eseguire alcun escape.

Stringhe

Ora, la domanda successiva è: come si inserisce un apostrofo in una stringa posta tra apostrofi? Dovreste già conoscere la risposta, o per essere precisi, due possibili risposte. Provate a stampare una stringa contenente il seguente messaggio:

I'm Monty Python.

Sapete come fare? .

Stringhe

Lo abbiamo già mostrato, ma vogliamo sottolineare ancora una volta questo fenomeno:

una stringa può essere vuota, cioè non contenere alcun carattere.

Una stringa vuota rimane comunque una stringa:

"

""

Valori booleani

Ogni volta che si chiede a Python se un numero è maggiore di un altro, la domanda porta alla creazione di un dato specifico, un valore **booleano**.

Il nome deriva da George Boole (1815-1864), autore dell'opera fondamentale *Le leggi del pensiero*, che contiene la definizione di **algebra booleana** - una parte dell'algebra che fa uso di due soli valori distinti: Vero e Falso, indicati come 1 e 0.

Un programmatore scrive un programma e il programma pone delle domande. Python esegue il programma e fornisce le risposte. Il programma deve essere in grado di reagire in base alle risposte ricevute.

Fortunatamente, i computer conoscono solo due tipi di risposte:

Sì, è vero;

No, questo è falso.

Non riceverete mai una risposta del tipo: *Non lo so o Probabilmente sì, ma non ne sono sicuro*.

Il pitone, quindi, è un rettile **binario**.

Questi due valori booleani hanno denotati in modo rigoroso in Python:

True

False

Non è possibile modificare nulla: questi simboli devono essere considerati così come sono, compresa **la sensibilità alle maiuscole**.

Valori booleani

Quale sarà l'output del seguente frammento di codice?

```
print(True > False)
```

```
print(True < False)
```

Eseguire il codice per verificare. Potete spiegare il risultato?

LAB-04

Tempo stimato: 5-10 minuti

Livello di difficoltà: Facile

Obiettivi

familiarizzare con la funzione print() e le sue capacità di formattazione;
esercitarsi nella codifica delle stringhe;
sperimentare il codice Python.

Scenario

Scrivete un pezzo di codice di una riga, utilizzando la funzione print() e i caratteri newline ed escape, in modo da far corrispondere il risultato atteso a tre righe.

Risultato previsto

```
"I'm"  
""learning""  
""Python""
```

None

Esiste un altro letterale speciale che viene utilizzato in Python: il letterale None. Questo letterale è un cosiddetto oggetto di tipo None e viene utilizzato per rappresentare **l'assenza di un valore**.

Esercizio 1

Quali sono i tipi di letterali dei due esempi seguenti?

"Ciao", "007"

Esercizio 2

Quali sono i tipi di letterali dei quattro esempi seguenti?

"1.5", 2.0, 528, Falso

Esercizio 3

Qual è il valore decimale del seguente numero binario?

1011

Operatori aritmetici: esponenziazione

Il segno `**` (doppio asterisco) è un operatore di **esponenziazione** (potenza). Il suo argomento di sinistra è la **base**, quello di destra l'**esponente**.

La matematica classica preferisce una notazione con apici, come questa: 2^3 . Gli editor di testo puri non lo accettano, quindi Python usa `**` al suo posto, ad esempio `2 ** 3`.

```
print(2 ** 3)
print(2 ** 3.)
print(2. ** 3)
print(2. ** 3.)
```

Gli esempi mostrano una caratteristica molto importante di quasi tutti gli **operatori numerici** di Python. Eseguite il codice e osservate attentamente i risultati che produce. Riuscite a vedere qualche regolarità?

Ricordate: è possibile formulare le seguenti regole sulla base di questo risultato:
quando **entrambi gli** argomenti di `**` sono interi, anche il risultato è un intero;
quando **almeno un** argomento di `**` è un float, anche il risultato è un float.

È una distinzione importante da ricordare.

Operatori aritmetici: moltiplicazione

Il segno * (asterisco) è un operatore di **moltiplicazione**.

Eseguite il codice sottostante e verificate se la regola degli *interi e dei float* funziona ancora.

```
print(2 * 3)
```

```
print(2 * 3.)
```

```
print(2. * 3)
```

```
print(2. * 3.)
```

Operatori aritmetici: divisione

Il segno / (barra) è un operatore **di divisione**.

Il valore davanti alla barra è un **dividendo**, quello dietro la barra un **divisore**.

Eseguite il codice sottostante e analizzate i risultati.

```
print(6 / 3)
```

```
print(6 / 3.)
```

```
print(6. / 3)
```

```
print(6. / 3.)
```

Dovreste vedere che c'è un'eccezione alla regola.

Il risultato prodotto dall'operatore di divisione è sempre un float, indipendentemente dal fatto che il risultato sembri o meno un float a prima vista: 1 / 2, o che sembri un puro numero intero: 2 / 1.

È un problema? Sì, lo è. A volte capita di aver bisogno di una divisione che fornisca un valore intero e non un float. Fortunatamente, Python può aiutarvi in questo senso.

Operatori aritmetici: divisione di interi

Il segno `//` (doppio slash) è un operatore **di divisione di interi**. Si differenzia dall'operatore standard `/` per due dettagli:

il suo risultato manca della parte frazionaria, che è assente (per gli interi) o è sempre uguale a zero (per i float); ciò significa che **i risultati sono sempre arrotondati**;

è conforme alla *regola dell'intero rispetto al float*.

Eseguire l'esempio seguente e vedere i risultati:

```
print(6 // 3)
```

```
print(6 // 3.)
```

```
print(6. // 3)
```

```
print(6. // 3.)
```

Come si può vedere, la *divisione intero per intero* dà un **risultato intero**. In tutti gli altri casi si ottengono risultati float.

Operatori aritmetici: divisione di interi

Eseguiamo alcuni test più avanzati.

Osservate il seguente frammento:

```
print(6 // 4)  
print(6. // 4)
```

Immaginate di usare / al posto di //: sapreste prevedere i risultati?

Sì, sarebbe 1,5 in entrambi i casi. È chiaro.

Ma quali risultati dobbiamo aspettarci con la // divisione?

Eseguite il codice e verificate voi stessi.

Si ottengono due numeri: uno intero e uno float.

Il risultato di una divisione intera viene sempre arrotondato al valore intero più vicino che è inferiore al risultato reale (non arrotondato).

Questo è molto importante: **l'arrotondamento va sempre al numero intero minore.**

Operatori aritmetici: divisione di interi

Guardate il codice qui sotto e provate a prevedere i risultati ancora una volta:

```
print(-6 // 4)
```

```
print(6. // -4)
```

Nota: alcuni valori sono negativi. Questo ovviamente influisce sul risultato. Ma come?

Il risultato è di due due negativi. Il risultato reale (non arrotondato) è -1,5 in entrambi i casi. Tuttavia, i risultati sono soggetti all'arrotondamento. L'**arrotondamento va verso il valore intero minore** e il valore intero minore è -2, quindi: -2 e -2,0.

NOTA

La divisione tra interi può anche essere chiamata **divisione per piani**.

Operatori: resto (modulo)

L'operatore successivo è piuttosto particolare, perché non ha un equivalente tra gli operatori aritmetici tradizionali.

La sua rappresentazione grafica in Python è il segno % (percentuale), che può sembrare un po' confuso.

Il risultato dell'operatore è il **resto lasciato dopo la divisione intera**.

In altre parole, è il valore che rimane dopo aver diviso un valore per un altro per ottenere un quoziente intero.

Nota: l'operatore è talvolta chiamato **modulo** in altri linguaggi di programmazione.

Guardate il seguente codice, cercate di prevedere il risultato e poi eseguitelo:

```
print(14 % 4)
```

Come si può vedere, il risultato è due. Ecco perché:

14 // 4 dà 3 → questo è il **quoziente** intero;

3 * 4 dà 12 → come risultato della **moltiplicazione di quoziente e divisore**;

14 - 12 dà 2 → questo è il **resto**.

Questo esempio è un po' più complicato:

```
print(12 % 4.5)
```

Qual è il risultato?

Operatori: come non dividere

Come probabilmente sapete, la **divisione per zero non funziona**.

Non provate a farlo:

- eseguire una divisione per zero;
- eseguire una divisione intera per zero;
- trovare il resto di una divisione per zero.

Operatori: addizione

L'operatore di **addizione** è il segno + (più), pienamente in linea con gli standard matematici.
Di nuovo, date un'occhiata al programma qui sotto:

```
print(-4 + 4)
```

```
print(-4. + 8)
```

Il risultato non dovrebbe essere sorprendente. Eseguite il codice per verificarlo.

L'operatore di sottrazione, gli operatori unari e binari

L'operatore di **sottrazione** è ovviamente il segno - (meno), anche se va notato che questo operatore ha anche un altro significato: **può cambiare il segno di un numero**.

Questa è una grande opportunità per presentare una distinzione molto importante tra operatori **unari** e **binari**. Nelle applicazioni di sottrazione, l'**operatore meno si aspetta due argomenti**: quello di sinistra (un **minuendo** in termini aritmetici) e quello di destra (un **sottraendo**).

Per questo motivo, l'operatore di sottrazione è considerato uno degli operatori binari, proprio come gli operatori di addizione, moltiplicazione e divisione.

Ma l'operatore meno può essere usato in un modo diverso (unario): date un'occhiata all'ultima riga qui sotto:

```
print(-4 - 4)
print(4. - 8)
print(-1.1)
```

A proposito: esiste anche un operatore unario +. Si può usare in questo modo:

```
print(+2)
```

L'operatore conserva il segno del suo unico argomento, quello destro.

Sebbene tale costruzione sia sintatticamente corretta, il suo utilizzo non ha molto senso e sarebbe difficile trovare una buona motivazione per farlo.

Gli operatori e le loro priorità

Si consideri la seguente espressione:

$$2 + 3 * 5$$

Probabilmente ricorderete a scuola che le **moltiplicazioni precedono le addizioni**.

Sicuramente ricorderete che dovrete prima moltiplicare 3 per 5 e, tenendo in memoria il 15, sommarlo a 2, ottenendo così il risultato di 17.

Il fenomeno che porta alcuni operatori ad agire prima di altri è noto come **gerarchia delle priorità**.

Python definisce con precisione le priorità di tutti gli operatori e presuppone che gli operatori con priorità maggiore eseguano le loro operazioni prima degli operatori con priorità minore.

Quindi, se si sa che `*` ha una priorità maggiore di `+`, il calcolo del risultato finale dovrebbe essere ovvio.

Operatori e loro vincoli

Il **binding** dell'operatore determina l'ordine dei calcoli eseguiti da alcuni operatori con uguale priorità, affiancati in un'espressione.

La maggior parte degli operatori di Python ha un binding sinistro, il che significa che il calcolo dell'espressione viene effettuato da sinistra a destra.

Questo semplice esempio vi mostrerà come funziona. Date un'occhiata:

```
print(9 % 6 % 2)
```

È possibile valutare questa espressione in due modi:

- da sinistra a destra: prima $9 \% 6$ dà 3, poi $3 \% 2$ dà 1;
- da destra a sinistra: prima $6 \% 2$ dà 0 e poi $9 \% 0$ causa **un errore fatale**.

Eseguire l'esempio e vedere cosa si ottiene.

Il risultato dovrebbe essere 1. Questo operatore è **vincolato a sinistra**. Ma c'è un'eccezione interessante.

Operatori e loro legami: esponenziazione

Ripetete l'esperimento, ma ora con l'esponenziazione.

Utilizzate questo frammento di codice:

```
print(2 ** 2 ** 3)
```

I due risultati possibili sono:

$2 ** 2 \rightarrow 4$; $4 ** 3 \rightarrow 64$

$2 ** 3 \rightarrow 8$; $2 ** 8 \rightarrow 256$

Eseguite il codice. Cosa si vede?

Il risultato mostra chiaramente che **l'operatore di esponenziazione utilizza il legame con il lato destro**.

Elenco delle priorità

La tabella seguente mostra una forma trunca delle priorità. Verrà ampliata in modo coerente con l'introduzione di nuovi operatori.

Priorità	Operatore
1	**
2	+, - (nota: gli operatori unari situati a destra dell'operatore di potenza si legano più fortemente)
3	*, /, //, %
4	+, -

Nota: abbiamo elencato gli operatori in ordine **di priorità, dalla più alta (1) alla più bassa (4)**.

Priorità

Cercate di elaborare la seguente espressione:

```
print(2 * 3 % 5)
```

Entrambi gli operatori (* e %) hanno la stessa priorità, quindi il risultato può essere indovinato solo se si conosce la direzione del legame. Cosa ne pensate? Qual è il risultato?

Operatori e parentesi

Naturalmente, è sempre consentito l'uso delle **parentesi**, che possono cambiare l'ordine naturale di un calcolo.

In base alle regole aritmetiche, le **sottoespressioni tra parentesi vengono sempre calcolate per prime**.

Si possono usare tutte le parentesi necessarie e spesso vengono utilizzate per **migliorare la leggibilità** di un'espressione, anche se non cambiano l'ordine delle operazioni.

Un esempio di espressione con più parentesi è riportato qui:

```
print((5 * ((25 % 13) + 100) / (2 * 13)) // 2)
```

Provate a calcolare il valore che viene stampato sulla console. Qual è il risultato della funzione print()?

Punti di forza

1. Un'**espressione** è una combinazione di valori (o variabili, operatori, richiami a funzioni, che imparerete presto a conoscere) che si valuta con un determinato valore, ad esempio $1 + 2$.
2. **Gli operatori** sono simboli speciali o parole chiave in grado di operare sui valori e di eseguire operazioni (matematiche), ad esempio, l'operatore $*$ moltiplica due valori: $x * y$.
3. Operatori aritmetici in Python: $+$ (addizione), $-$ (sottrazione), $*$ (moltiplicazione), $/$ (divisione classica - restituisce sempre un float), $\%$ (modulo - divide l'operando di sinistra per quello di destra e restituisce il resto dell'operazione, ad es, $5 \% 2 = 1$), $**$ (esponenziazione - l'operando di sinistra viene elevato alla potenza dell'operando di destra, ad esempio, $2 ** 3 = 2 * 2 * 2 = 8$), $//$ (divisione intera/integrale - restituisce un numero risultante dalla divisione, ma arrotondato per difetto al numero intero più vicino, ad esempio, $3 // 2,0 = 1,0$)
4. Un operatore **unario** è un operatore con un solo operando, ad esempio -1 , o $+3$.
5. Un operatore **binario** è un operatore con due operandi, ad esempio $4 + 5$ oppure $12 \% 5$.
6. Alcuni operatori agiscono prima di altri - **la gerarchia delle priorità**:
l'operatore $**$ (esponenziazione) ha la massima priorità;
poi gli operatori unari $+$ e $-$ (nota: un operatore unario a destra dell'operatore di esponenziazione si lega più fortemente, ad esempio: $4 ** -1$ uguale $0,25$)
poi $*$, $/$, $//$ e $\%$;
e, infine, la priorità più bassa: i binari $+$ e $-$.
7. Le sottoespressioni tra **parentesi** vengono sempre calcolate per prime, ad esempio, $15 - 1 * (5 * (1 + 2)) = 0$.
8. L'operatore di **esponenziazione** usa la **legatura a destra**, ad esempio $2 ** 2 ** 3 = 256$.

Punti di forza

1. Un'**espressione** è una combinazione di valori (o variabili, operatori, richiami a funzioni, che imparerete presto a conoscere) che si valuta con un determinato valore, ad esempio $1 + 2$.
2. **Gli operatori** sono simboli speciali o parole chiave in grado di operare sui valori e di eseguire operazioni (matematiche), ad esempio, l'operatore $*$ moltiplica due valori: $x * y$.
3. Operatori aritmetici in Python: $+$ (addizione), $-$ (sottrazione), $*$ (moltiplicazione), $/$ (divisione classica - restituisce sempre un float), $\%$ (modulo - divide l'operando di sinistra per quello di destra e restituisce il resto dell'operazione, ad es, $5 \% 2 = 1$), $**$ (esponenziazione - l'operando di sinistra viene elevato alla potenza dell'operando di destra, ad esempio, $2 ** 3 = 2 * 2 * 2 = 8$), $//$ (divisione intera/integrale - restituisce un numero risultante dalla divisione, ma arrotondato per difetto al numero intero più vicino, ad esempio, $3 // 2,0 = 1,0$)
4. Un operatore **unario** è un operatore con un solo operando, ad esempio -1 , o $+3$.
5. Un operatore **binario** è un operatore con due operandi, ad esempio $4 + 5$ oppure $12 \% 5$.
6. Alcuni operatori agiscono prima di altri - **la gerarchia delle priorità**:
l'operatore $**$ (esponenziazione) ha la massima priorità;
poi gli operatori unari $+$ e $-$ (nota: un operatore unario a destra dell'operatore di esponenziazione si lega più fortemente, ad esempio: $4 ** -1$ uguale $0,25$)
poi $*$, $/$, $//$ e $\%$;
e, infine, la priorità più bassa: i binari $+$ e $-$.
7. Le sottoespressioni tra **parentesi** vengono sempre calcolate per prime, ad esempio, $15 - 1 * (5 * (1 + 2)) = 0$.
8. L'operatore di **esponenziazione** usa la **legatura a destra**, ad esempio $2 ** 2 ** 3 = 256$.

Esercizio 1

Qual è l'output del seguente snippet?

```
print((2 ** 4), (2 * 4.), (2 * 4))
```

Esercizio 2

Qual è l'output del seguente snippet?

```
print((-2 / 4), (2 / 4), (2 // 4), (-2 // 4))
```

Esercizio 3

Qual è l'output del seguente snippet?

```
print((2 % -4), (2 % 4), (2 ** 3 ** 2))
```

Cosa sono le variabili?

Le variabili di Python hanno:

- un nome;
- un valore (il contenuto del contenitore)

Cominciamo con i problemi legati al nome di una variabile.

Se si vuole **dare un nome a una variabile**, è necessario seguire alcune regole precise:

- il nome della variabile deve essere composto da lettere maiuscole o minuscole, cifre e dal carattere _ (underscore)
- il nome della variabile deve iniziare con una lettera;
- il carattere _ è una lettera;
- Le lettere maiuscole e minuscole sono trattate in modo diverso
- il nome della variabile non deve essere una delle parole riservate di Python

Cosa sono le variabili?

La [PEP 8 -- Guida allo stile del codice Python](#) raccomanda la seguente convenzione di denominazione per le variabili e le funzioni in Python:

- I nomi delle variabili devono essere minuscoli, con le parole separate da trattini bassi per migliorare la leggibilità (ad esempio, var, my_variable).
- I nomi delle funzioni seguono la stessa convenzione dei nomi delle variabili (ad esempio, fun, my_function).
- è anche possibile usare casi misti (ad esempio, myVariable), ma solo in contesti in cui questo è già lo stile prevalente, per mantenere la compatibilità con la convenzione adottata

Parole chiave

Date un'occhiata all'elenco di parole che svolgono un ruolo molto particolare in ogni programma Python.

['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield'].

Si chiamano **parole chiave** o (più precisamente) **parole chiave riservate**. Sono riservate perché **non devono essere usate come nomi**: né per le variabili, né per le funzioni, né per qualsiasi altra entità denominata che si voglia creare.

Il significato della parola riservata è **predefinito** e non deve essere modificato in alcun modo.

Fortunatamente, grazie al fatto che Python è sensibile alle maiuscole e alle minuscole, è possibile modificare una qualsiasi di queste parole cambiando il caso di una qualsiasi lettera, creando così una nuova parola, che non è più riservata.

Ad esempio, **non si può dare un nome** alla variabile come questo:

```
import
```

Non si può avere una variabile denominata in questo modo: è proibito. Ma si può fare così:

```
Import
```

Creazione di variabili

Cosa si può mettere dentro una variabile?

Qualsiasi cosa.

È possibile utilizzare una variabile per memorizzare qualsiasi valore di uno dei tipi già presentati e molti altri non ancora mostrati.

Il valore di una variabile è quello che avete inserito in essa. Può variare ogni volta che se ne ha bisogno o che si vuole. Può essere un numero intero un momento, e un float un momento dopo, per poi diventare una stringa.

Parliamo ora di due cose importanti: **come si creano le variabili** e **come si inseriscono i valori al loro interno** (o meglio, come si danno o si **passano i valori** alle variabili).

RICORDA

Una variabile nasce come risultato dell'assegnazione di un valore ad essa. A differenza di altri linguaggi, non è necessario dichiararla in modo particolare.

Se si assegna un valore a una variabile inesistente, la variabile viene **creata automaticamente**. Non è necessario fare altro.

La creazione (o altrimenti - la sua sintassi) è estremamente semplice: **basta usare il nome della variabile desiderata, poi il segno di uguale (=) e il valore che si vuole inserire nella variabile.**

Date un'occhiata al seguente codice:

```
var = 1  
print(var)
```

Consiste in due semplici istruzioni:

La prima crea una variabile di nome var e le assegna un letterale con un valore intero pari a 1.

Il secondo stampa il valore della variabile appena creata nella console.

Utilizzo delle variabili

È possibile utilizzare tutte le dichiarazioni di variabili necessarie per raggiungere il proprio obiettivo, come in questo caso:

```
var = 1
```

```
saldo_conto = 1000.0
```

```
nome_cliente = 'John Doe'
```

```
print(var, conto_bilancio, nome_cliente)
```

```
print(var)
```


Utilizzo delle variabili

Non è consentito utilizzare una variabile che non esiste (in altre parole, una variabile a cui non è stato assegnato un valore).

Questo esempio **causerà un errore**:

```
var = 1  
print(Var)
```

Abbiamo provato a usare una variabile chiamata Var, che non ha alcun valore (nota: var e Var sono entità diverse e non hanno nulla in comune per quanto riguarda Python).

RICORDA

È possibile utilizzare la funzione print() e combinare testo e variabili utilizzando l'operatore + per produrre stringhe e variabili, ad es:

```
var = "3.8.5"  
print("Versione di Python: " + var)
```

Assegnazione di un nuovo valore a una variabile già esistente

Come si assegna un nuovo valore a una variabile già creata? Allo stesso modo. Basta usare il segno di uguale.

Il segno di uguale è in realtà un **operatore di assegnazione**. Anche se può sembrare strano, l'operatore ha una sintassi semplice e un'interpretazione univoca.

Assegna il valore del suo argomento destro a quello sinistro, mentre l'argomento destro può essere un'espressione arbitrariamente complessa che coinvolge letterali, operatori e variabili già definite.

Guardate il codice qui sotto:

```
var = 1  
print(var)  
var = var + 1  
print(var)
```

Il codice invia due righe alla console:

```
1  
2
```

Python tratta il segno `=` non come *uguale a*, ma come *assegnazione di un valore*

Risolvere semplici problemi matematici

Ora dovrete essere in grado di costruire un breve programma che risolva semplici problemi matematici come il teorema di Pitagora:

Il quadrato dell'ipotenusa è uguale alla somma dei quadrati degli altri due lati.

Il codice seguente valuta la lunghezza dell'ipotenusa utilizzando il teorema di Pitagora:

```
a = 3.0
b = 4.0
c = (a ** 2 + b ** 2) ** 0,5
print("c =", c)
```

Nota: è necessario utilizzare l'operatore `**` per valutare la radice quadrata come:

$$\sqrt{x} = x^{(1/2)}$$

e

$$c = \sqrt{a^2 + b^2}$$

Riuscite a indovinare l'output del codice?

LAB-05

Tempo stimato: 10 minuti

Livello di difficoltà: Facile

Obiettivi

acquisire familiarità con il concetto di memorizzazione e lavoro con diversi tipi di dati in Python; sperimentare il codice Python.

Scenario

Ecco una breve storia:

C'era una volta, nel Paese delle mele, John con tre mele, Mary con cinque mele e Adam con sei mele. Erano tutti molto felici e vissero a lungo. Fine della storia.

Il vostro compito è quello di:

- creare le variabili: john, mary e adam;
- assegnare valori alle variabili. I valori devono essere uguali al numero di frutti posseduti rispettivamente da Giovanni, Maria e Adamo;
- dopo aver memorizzato i numeri nelle variabili, stampare le variabili su una riga e separare ciascuna di esse con una virgola;
- creare ora una nuova variabile chiamata total_apples pari alla somma delle tre variabili precedenti.
- stampa nella console il valore memorizzato in total_apples;
- Sperimentate con il vostro codice: create nuove variabili, assegnate loro valori diversi ed eseguite varie operazioni aritmetiche su di esse (ad esempio, +, -, *, /, //, ecc.). Provate a stampare una stringa e un numero intero insieme su una riga, ad esempio "Numero totale di mele:" e total_apples.

Operatori Shortcut

Molto spesso si vuole utilizzare una stessa variabile sia a destra che a sinistra dell'operatore =.

Ad esempio, se dobbiamo calcolare una serie di valori successivi di potenze di 2, oppure incrementare di uno una variabile:

```
x = x * 2
```

```
pecora = pecora + 1
```

Python offre un modo abbreviato di scrivere operazioni come queste, che possono essere codificate come segue:

```
x *= 2
```

```
pecora += 1
```

Se op è un operatore a due argomenti (condizione molto importante) e l'operatore viene utilizzato nel seguente contesto:

```
variabile = espressione op variabile
```

Si può semplificare e mostrare come segue:

```
variabile op= espressione
```

Operatori Shortcut

Date un'occhiata agli esempi che seguono. Assicuratevi di averli compresi tutti.

$i = i + 2 * j \Rightarrow i += 2 * j$

$var = var / 2 \Rightarrow var /= 2$

$rem = rem \% 10 \Rightarrow rem \% = 10$

$j = j - (i + var + rem) \Rightarrow j -= (i + var + rem)$

$x = x ** 2 \Rightarrow x ** = 2$

LAB-06

Tempo stimato: 10 minuti

Livello di difficoltà: Facile

Obiettivi

familiarizzare con il concetto di variabile e lavorare con essa;

eseguire calcoli e conversioni di base;

sperimentare il codice Python.

Scenario

Miglia e chilometri sono unità di misura della lunghezza o della distanza.

Tenendo presente che 1 miglio equivale a circa 1,61 chilometri, completare il programma nell'editor in modo che si converta:

miglia a chilometri;

chilometri a miglia.

Non modificate nulla del codice esistente. Scrivete il codice nei punti indicati da `###`. Testate il vostro programma con i dati forniti nel codice sorgente.

LAB-06

Codice:

```
chilometri = 12,25
```

```
miglia = 7,38
```

```
miglia_a_chilometri = ###
```

```
chilometri_a_miglia = ###
```

```
print(miglia, "miglia è", round(miglia_a_chilometri, 2), "chilometri")
```

```
print(chilometri, "chilometri è", round(chilometri_a_miglia, 2), "miglia")
```

Risultato previsto

7,38 miglia sono 11,88 chilometri

12,25 chilometri sono 7,61 miglia

Prestate particolare attenzione a ciò che accade all'interno della funzione `print()`. Analizzate come forniamo più argomenti alla funzione e come produciamo i dati attesi.

Si noti che alcuni degli argomenti della funzione `print()` sono stringhe (ad esempio, "miles is"), mentre altri sono variabili (ad esempio, miles).

LAB-06

CONSIGLIO

C'è un'altra cosa interessante che sta accadendo. Riuscite a vedere un'altra funzione all'interno della funzione `print()`? È la funzione **`round()`**. Il suo compito è quello di arrotondare il risultato in uscita al numero di decimali specificato tra le parentesi e di restituire un float (all'interno della funzione `round()` si trovano il nome della variabile, una virgola e il numero di decimali che vogliamo ottenere).

Dopo aver completato il laboratorio, fate altri esperimenti. Provate a scrivere diversi convertitori, ad esempio un convertitore da USD a EUR, un convertitore di temperatura, ecc. Provate a utilizzare e sperimentare la funzione `round()` per arrotondare i risultati a uno, due o tre decimali. Verificate cosa succede se non fornite alcun numero di cifre.

LAB-07

Tempo stimato: 10-15 minuti

Livello di difficoltà: Facile

Obiettivi

familiarizzare con il concetto di numeri, operatori e operazioni aritmetiche in Python;
eseguire calcoli di base.

Scenario

Osservate il codice seguente:

```
x = # codifica in modo rigido i dati del test
x = float(x)
# scrivete il vostro codice qui
print("y =", y)
```

legge un valore float, lo inserisce in una variabile di nome x e stampa il valore di una variabile di nome y. Il vostro compito è completare il codice per valutare la seguente espressione:

$$3x^3 - 2x^2 + 3x - 1$$

Il risultato deve essere assegnato a y.

Notate come cambiamo il tipo di dati per assicurarci che x sia di tipo float.

LAB-07

Mantenete il vostro codice pulito e leggibile e testatelo utilizzando i dati che vi abbiamo fornito, assegnandoli ogni volta alla variabile x (codificandola in modo fisso).

Dati del test

Ingresso campione

$x = 0$

$x = 1$

$x = -1$

Risultati attesi

$y = -1.0$

$y = 3.0$

$y = -9.0$

Esercizio 1

Quali dei seguenti nomi di variabili sono illegali in Python?

my_var

m

101

nome variabile medio-lungo

m101

m 101

Del

del

Esercizio 2

Qual è l'output del seguente snippet?

a = '1'

b = "1"

stampa(a + b)

Esercizio 3

Qual è l'output del seguente snippet?

```
a = 6
```

```
b = 3
```

```
a /= 2 * b
```

```
print(a)
```

Lasciare i commenti nel codice: perché, come e quando?

Il commento è completamente trasparente: dal punto di vista di Python, si tratta solo di uno spazio (indipendentemente dalla lunghezza del commento reale).

In Python, un commento è un pezzo di testo che inizia con un segno # (hash) e si estende fino alla fine della riga. Se si vuole un commento che si estende su più righe, bisogna mettere un hash davanti a tutte.

Proprio come qui:

```
# Questo programma valuta l'ipotenusa c.
```

```
# a e b sono le lunghezze delle gambe.
```

```
a = 3.0
```

```
b = 4.0
```

```
c = (a ** 2 + b ** 2) ** 0,5 # Usiamo ** invece della radice quadrata.
```

```
print("c =", c)
```

Esercizio 1

Qual è l'output del seguente snippet?

```
# print("Stringa #1")  
print("Stringa #2")
```

Esercizio 2

Cosa succede quando si esegue il seguente codice?

```
# Questo è  
una linea multipla  
commento. #
```

```
print("Ciao!")
```

La funzione input()

La funzione input() è in grado di leggere i dati inseriti dall'utente e di restituire gli stessi dati al programma in esecuzione.

Il programma può manipolare i dati, rendendo il codice veramente interattivo.

```
print("Tell me anything...")
anything = input()
print("Hmm...", anything, "... Really?")
```

Nota:

- Il programma chiede all'utente di immettere alcuni dati dalla console
- la funzione input() viene invocata senza argomenti (questo è il modo più semplice di usare la funzione); la funzione commuta la console in modalità di input; si vedrà un cursore lampeggiante e si potranno inserire alcuni tasti, terminando con il tasto *Invio*; tutti i dati inseriti saranno inviati al programma attraverso il risultato della funzione;
- nota: è necessario assegnare il risultato a una variabile; questo è un passaggio cruciale: se lo si tralascia, i dati inseriti andranno persi;
- quindi utilizziamo la funzione print() per visualizzare i dati ottenuti, con alcune osservazioni aggiuntive.

La funzione `input()` con un argomento

La funzione `input()` può fare anche qualcos'altro: può richiedere all'utente di essere contattato senza l'aiuto di `print()`.

Abbiamo modificato un po' il nostro esempio, guardate il codice:

```
anything = input("Tell me anything...")  
print("Hmm...", anything, "...Really?")
```

Nota:

- la funzione `input()` viene invocata con un solo argomento: una stringa contenente un messaggio;
- il messaggio verrà visualizzato sulla console prima che l'utente abbia la possibilità di inserire qualcosa;
- `input()` farà il suo lavoro.

Questa variante dell'invocazione `input()` semplifica il codice e lo rende più chiaro.

Il risultato della funzione input()

L'abbiamo già detto, ma va ribadito senza ambiguità: il risultato della funzione input() è una stringa. Una stringa contenente tutti i caratteri immessi dall'utente dalla tastiera. Non è un intero o un float. Ciò significa che non è possibile utilizzarlo come argomento di alcuna operazione aritmetica, ad esempio non si può usare questo dato per elevarlo al quadrato, dividerlo per qualcosa o dividere qualcosa per esso.

```
anything = input("Enter a number: ")  
something = anything ** 2.0  
print(anything, "to the power of 2 is", something)
```

La funzione input() - operazioni non consentite

Guardate il seguente codice:

Testing TypeError message.

```
anything = input("Enter a number: ")
something = anything ** 2.0
print(anything, "to the power of 2 is", something)
```

Eseguite, inserite un numero qualsiasi e premete *Invio*.

Cosa succede?

Python dovrebbe fornire il seguente risultato:

Traceback (most recent call last):

File ".main.py", line 4, in <module>

```
something = anything ** 2.0
```

TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'float'

L'ultima riga della frase spiega tutto: avete cercato di applicare l'operatore ** tra 'str' (stringa) e 'float'.

Questo è vietato.

Casting

Python offre due semplici funzioni per specificare un tipo di dati e risolvere questo problema: `int()` e `float()`. I loro nomi si commentano da soli:

- la funzione `int()` **prende un argomento** (ad esempio, una stringa: `int(stringa)`) e cerca di convertirlo in un numero intero; se fallisce, fallisce anche l'intero programma.
- La funzione `float()` prende un argomento (ad esempio, una stringa: `float(stringa)`) e cerca di convertirlo in un float

È molto semplice e molto efficace. Inoltre, è possibile richiamare qualsiasi funzione passando direttamente i risultati di `input()`. Non è necessario utilizzare alcuna variabile come memoria intermedia.

Abbiamo implementato l'idea nell'editor: date un'occhiata al codice.

```
anything = float(input("Enter a number: "))  
something = anything ** 2.0  
print(anything, "to the power of 2 is", something)
```

Provate a eseguire il codice modificato. Non dimenticate di inserire un **numero valido**.

Controllate alcuni valori diversi, piccoli e grandi, negativi e positivi. Anche lo zero è un buon input.

La funzione input()

Il prossimo esempio si riferisce al programma precedente per trovare la lunghezza dell'ipotenusa. Riscriviamolo e rendiamolo in grado di leggere le lunghezze dei lati dalla console

```
leg_a = float(input("Input first leg length: "))
```

```
leg_b = float(input("Input second leg length: "))
```

```
print("Hypotenuse length is", (leg_a**2 + leg_b**2) **.5)
```

Eseguitelo e provate a inserire alcuni valori negativi.

Il programma, purtroppo, non reagisce a questo errore evidente.

Per ora ignoriamo questa debolezza.