

# Il Tutorial di Python

Python è un linguaggio di programmazione potente e di facile apprendimento. Utilizza efficienti strutture dati di alto livello e un semplice ma efficace approccio alla programmazione orientata agli oggetti.

# Caratteristiche Principali di Python

## Sintassi Elegante

L'elegante sintassi di Python e la tipizzazione dinamica lo rendono ideale per lo scripting e lo sviluppo rapido di applicazioni.

## Natura Interpretata

Come linguaggio interpretato, Python offre flessibilità e facilità di debug durante lo sviluppo.

## Multipiattaforma

Disponibile per tutte le principali piattaforme, garantendo portabilità del codice.

The screenshot shows a Jupyter Notebook interface with three code cells. A red arrow points from the text 'with basic operations' in the first cell to the `__init__` method in the code. The first cell is titled '1: Class Definition - with basic operations'. The second cell is titled '2: Class Definition - additional functions'. The third cell is titled '3: Creating & Executing Objects'. Each cell contains Python code and its execution output.

**Cell 1: Class Definition - with basic operations**

```
1 import math
2
3 class Calc:
4     def __init__(self):
5         self.result = 0
6
7     def add(self, a, b):
8         """Add two numbers."""
9         self.result = a + b
10        return self.result
11
12    def subtract(self, a, b):
13        """Subtract b from a."""
14        self.result = a - b
15        return self.result
16
```

Command took 0.18 seconds -- by 14.3 at 8/16/2024, 3:21:43 PM on Shared

**Cell 2: Class Definition - additional functions**

```
1 class Calc(Calc):
2     def power(self, base, exponent):
3         """Calculate base raised to the power of exponent."""
4         self.result = math.pow(base, exponent)
5         return self.result
6
7     def square_root(self, num):
8         """Calculate the square root of a number."""
9         if num < 0:
10             raise ValueError("Cannot calculate square root of a negative number")
11         self.result = math.sqrt(num)
12         return self.result
13
14     def factorial(self, num):
15         """Calculate the factorial of a number."""
16         if num < 0:
17             raise ValueError("Cannot calculate factorial of a negative number")
18         self.result = math.factorial(num)
19         return self.result
```

Command took 0.08 seconds -- by 14.3 at 8/16/2024, 3:21:43 PM on Shared

**Cell 3: Creating & Executing Objects**

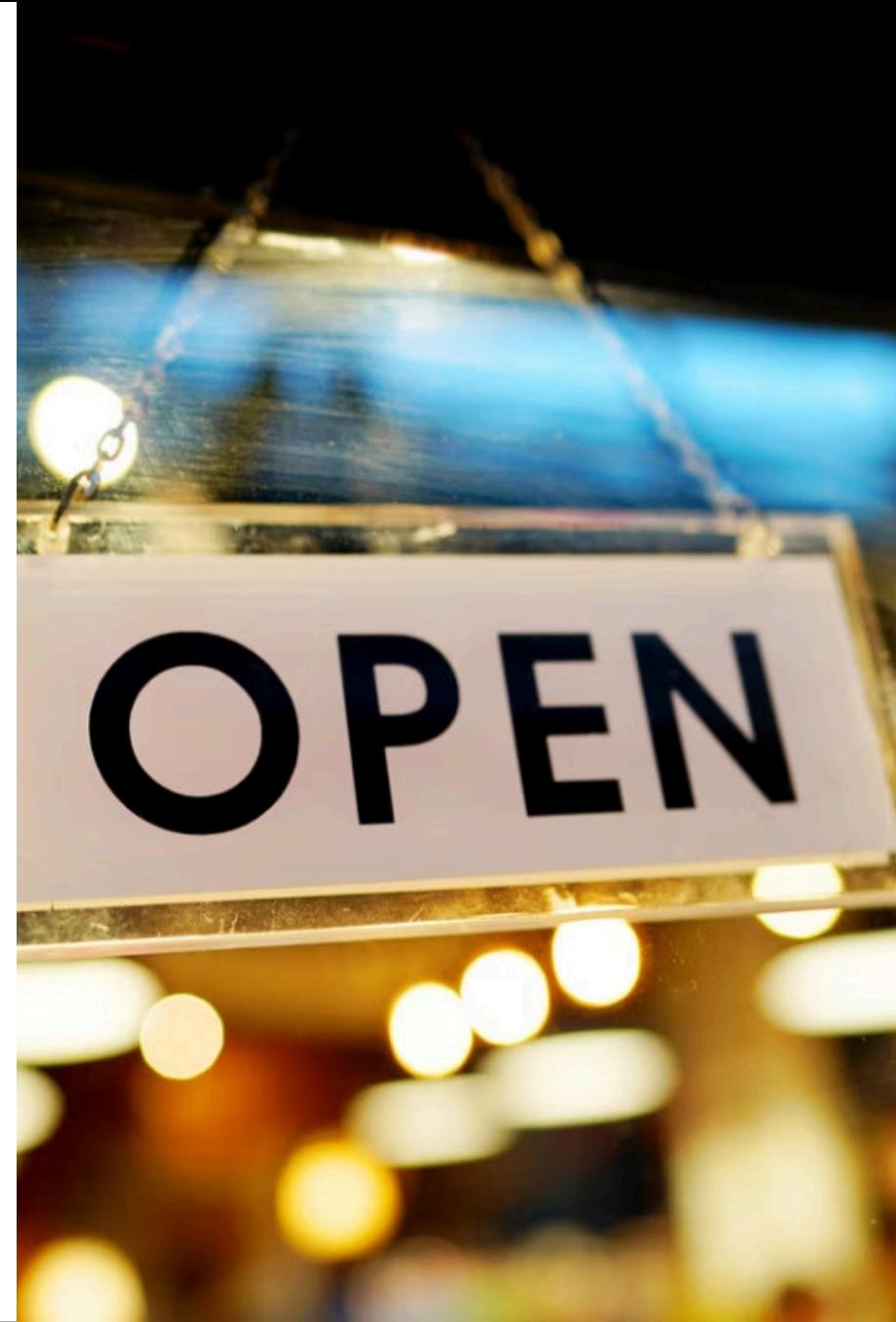
```
1 # Create an instance of the Calc class
2 calculator = Calc()
3
4 # Perform calculations
5 print("Add ", calculator.add(5, 5))
6 print("Sub ", calculator.subtract(6, 7))
7 print("Pow ", calculator.power(2, 3))
8 print("Sq Rt ", calculator.square_root(121))
9 print("Fac ", calculator.factorial(7))
```

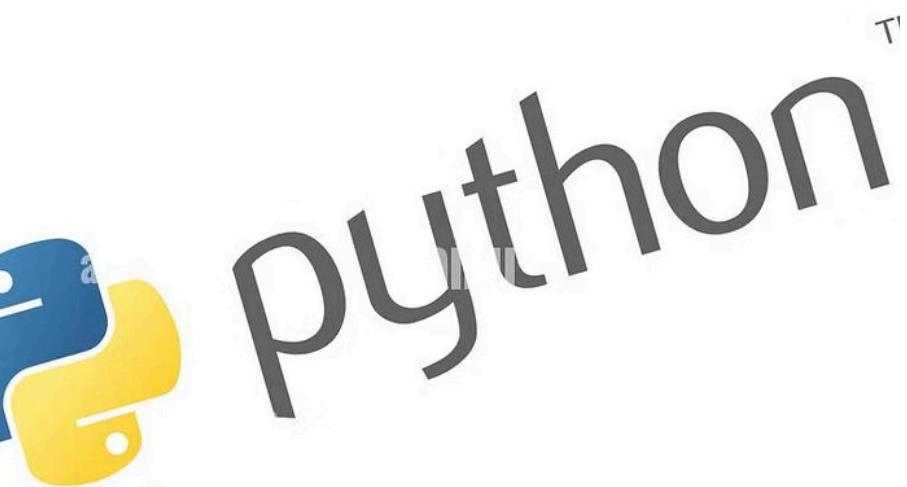
Add 10  
Sub -1  
Pow 8.0  
Sq Rt 11.0  
Fac 5040

# Disponibilità e Distribuzione

L'interprete Python e l'ampia libreria standard sono liberamente disponibili in file sorgenti o binari per tutte le principali piattaforme sul sito web di Python.

Il sito contiene anche puntatori a molti moduli Python liberi e gratuiti di terzi, interi programmi, strumenti di sviluppo e documentazione addizionale.





# Estensibilità e Personalizzazione

## Estensioni in C/C++

L'interprete Python è facilmente estendibile con nuove funzioni o tipi di dato implementati in C o C++.

## Linguaggio di Estensione

Python è anche adatto come linguaggio di estensione per applicazioni personalizzabili.

# LEARNING PYTHON

CRASH COURSE TUTORIAL



GUIDO VAN ROSSUM

## Obiettivi del Tutorial

Questo tutorial introduce informalmente il lettore ai concetti e alle caratteristiche base del linguaggio e del sistema Python. È di aiuto avere un interprete Python a portata di mano per fare esperienza diretta.

Tutti gli esempi sono autoesplicativi, quindi il tutorial può essere letto anche a elaboratore spento.

Prof. Fedeli Mass

# Risorse Aggiuntive

01

## **La Libreria di Riferimento**

Per una descrizione degli oggetti e dei moduli standard

02

## **Il Manuale di Riferimento**

Fornisce una definizione più formale del linguaggio

03

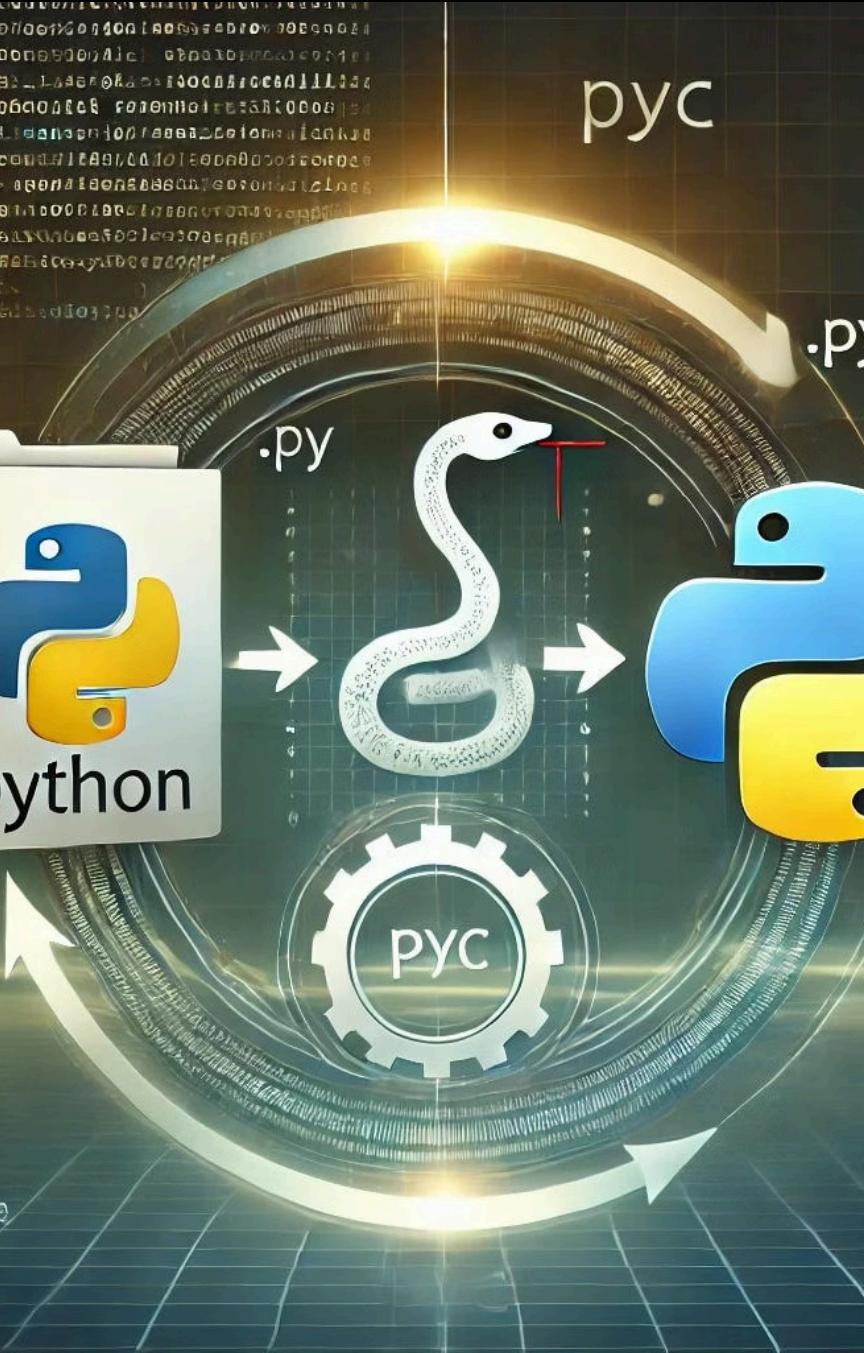
## **Guide per Estensioni**

Per scrivere estensioni in C o C++

04

## **Libri Specializzati**

Numerosi libri che si occupano in modo approfondito di Python



# Filosofia del Tutorial

Questo tutorial non si propone di essere onnicomprensivo e di coprire ogni singola funzionalità. Vuole essere piuttosto un'introduzione alle caratteristiche più notevoli di Python e fornire un'idea precisa dello stile del linguaggio.

Dopo averlo letto si sarà capaci di leggere e scrivere moduli e programmi in Python, e quindi pronti ad imparare di più sui vari moduli della libreria Python.

# Usare l'Interprete Python

L'interprete Python sulle macchine UNIX è di solito installato in '/usr/local/bin/'. Aggiungendo questa directory al percorso di ricerca della shell è possibile farlo partire digitando il comando 'python'.

# Avvio e Uscita dall'Interprete

## Carattere di EOF

Digitare Control-D su UNIX o Control-Z su Windows al prompt primario fa sì che l'interprete esca con status zero.

## Comando di Uscita

Se non funziona, si può uscire digitando: 'import sys; sys.exit()'

## History

Ctrl-p	Fetch the previous command from the history list.
Ctrl-n	Fetch the next command from the history list.
Alt-<	Move to the first line in the history.
Alt->	Move to the last line in the history.
Ctrl-r	Search backward through history.
Ctrl-s	Search forward through history.
Alt-p	Search backward through history for a given string.
Alt-n	Search forward through history for a given string.
Ctrl-Alt-y	Insert the first argument to the previous command. With an argument n, insert the nth word from the previous command.
Alt-.	Insert the last argument to the previous command. With an argument n, insert the nth word from the previous command.
Alt-_	

Alt-y

Rotate the kill ring, and yank (paste) the new top. Only works after a yank.

## Completing

Tab	Autocomplete.
Alt-?	List possible completions.
Alt-*	Insert possible completions.

## Macros

Ctrl-x (	Begin saving the characters typed as a macro.
Ctrl-x )	Stop saving the characters typed as a macro.
Ctrl-x e	Execute the most recent macro.

# Funzionalità di Editing

Le funzioni di editing di riga dell'interprete di solito non sono molto sofisticate. Su UNIX, chiunque abbia installato l'interprete può avere abilitato il supporto per la libreria GNU readline.

Questa aggiunge funzionalità di storico e di editing interattivo più avanzate. Il modo più rapido per controllare se sia supportato l'editing è digitare Control-P al primo prompt Python.

# Modalità di Esecuzione



## Modalità Interattiva

1

L'interprete legge ed esegue interattivamente dei comandi quando lo standard input è connesso ad un terminale

## Esecuzione Script

2

Quando invocato con un nome di file come argomento, legge ed esegue uno script da quel file

## Comando Diretto

3

Con 'python -c comando' esegue le istruzioni contenute nel comando

# Passaggio di Argomenti

Quando noti all'interprete, il nome dello script e gli argomenti addizionali sono passati allo script tramite la variabile `sys.argv`, che è una lista di stringhe.

La sua lunghezza minima è uno. Quando non vengono forniti né script né argomenti, `sys.argv[0]` è una stringa vuota.

```
umentParser(description='Authenticate user with u  
--username', '-u', type=str, required=True, help=  
--password', '-p', type=str, required=True, help=  
rgs()  
  
orm authentication using the provided username and  
s.username, args.password)  
  
rname, password):  
    logic  
        and password == 'admin123':  
            tion successful!")  
  
tion failed. Please check your credentials.")  
:  
:
```

```
{1, 2, 3, 4}
[>>> for i in mySet:
[... print(i)
  File "<stdin>", line 2
    print(i)
    ^
IndentationError: expected an indented block
[>>> history(3)
```

## Modalità Interattiva

Quando i comandi vengono letti da un terminale, si dice che l'interprete è in modalità interattiva. In questa modalità presenta un prompt primario ('>>> ') e per le righe di continuazione il prompt secondario ('... ').

L'interprete stampa un messaggio di benvenuto con il numero di versione e un avviso di copyright prima del prompt iniziale.

# Gestione degli Errori

Quando sopreviene un errore, l'interprete stampa un messaggio di errore e una traccia dello stack. Se si trova in modalità interattiva ritorna poi al prompt primario.

Se l'input proveniva da un file, esce con uno stato diverso da zero dopo aver stampato la traccia dello stack.

```
1 import functools
2
3 def catch_all_and_print(f):
4     # type: (Callable[..., Any]) -> Callable[..., Any]
5     """
6         A function wrapper for catching all exceptions and logging them
7     """
8     @functools.wraps(f)
9     def inner(*args, **kwargs):
10         # type: (*Any, **Any) -> Any
11         try:
12             return f(*args, **kwargs)
13         except Exception as ex:
14             print(ex)
15
16     return inner
17
18 @catch_all_and_print
19 def my_method():
20     raise Exception("hello world")
21
22 my_method()
23
24 # output:
25 # "hello world"
26
```

catch\_all\_error\_handler.py hosted with ❤ by GitHub

[view raw](#)

# Interruzioni e Eccezioni

## Interruzione da Tastiera

Digitando il carattere di interruzione (di solito Ctrl-C) al prompt si cancella l'input e si ritorna al prompt primario.

## Eccezione KeyboardInterrupt

Digitando un'interruzione mentre un comando è in esecuzione viene sollevata un'eccezione KeyboardInterrupt, gestibile tramite try.



# Script Python Eseguibili

Sui sistemi UNIX in stile BSD, gli script Python possono essere resi direttamente eseguibili ponendo all'inizio dello script la riga:

```
#!/usr/bin/env python
```

I caratteri '#!' devono essere i primi due del file. Lo script può essere reso eseguibile usando il comando chmod:

```
$ chmod +x myscript.py
```

```
9 # ISO 8859-2 (Latin-2), used for Central European languages
10 encoding = 'iso8859-2'
11 character = bytes([code_point]).decode(encoding)
12 print(f"Character code {code_point} in {encoding} (Central European): '{character}'")
13
14 # ISO 8859-5 (Cyrillic), used for Cyrillic alphabets
15 encoding = 'iso8859-5'
16 character = bytes([code_point]).decode(encoding)
17 print(f"Character code {code_point} in {encoding} (Cyrillic alphabets): '{character}'")
18
19 # ISO 8859-7 (Greek), used for the Greek language
20 encoding = 'iso8859-7'
21 character = bytes([code_point]).decode(encoding)
```



Outputs different characters for the ASCII code 224 depending on the encoding we used.

## Codifica dei File Sorgenti

È possibile usare una codifica differente dall'ASCII nei file sorgenti Python. La strada maestra consiste nell'inserire una speciale riga di commento che definisca la codifica del file sorgente:

```
# -*- coding: iso-8859-1 -*-
```

Con questa dichiarazione, tutti i caratteri nel sorgente verranno elaborati come iso-8859-1 e sarà possibile scrivere direttamente stringhe costanti Unicode nella codifica selezionata.

```
'model_path': '\\models\\Qwen\\Qwen-VL-Chat-Int4',
'port': 5554}
当前Embeddings模型: m3e-base @ cuda
=====Langchain-Chatchat Configuration=====

2023-10-27 16:17:34 | INFO | root | 正在启动服务:
2023-10-27 16:17:34 | INFO | root | 如需查看 llm_api 日志, 请前往 F:\Langchain-Chatchat\logs
2023-10-27 16:17:40 | ERROR | stderr | INFO: Started server process [16472]
2023-10-27 16:17:40 | ERROR | stderr | INFO: Waiting for application startup.
2023-10-27 16:17:40 | ERROR | stderr | INFO: Application startup complete.
2023-10-27 16:17:40 | ERROR | stderr | INFO: Uvicorn running on http://127.0.0.1:5553 (Press CTRL+C to quit)
Loading checkpoint shards: 100%|██████████| 5/5 [00:00<00:00, 10.85it/s]
```

# File di Avvio Interattivo

Quando si usa Python interattivamente, risulta spesso comodo che alcuni comandi standard vengano eseguiti ad ogni lancio dell'interprete.

È possibile farlo configurando una variabile d'ambiente chiamata PYTHONSTARTUP con il nome di un file contenente i propri comandi di avvio.

# Un'Introduzione Informale a Python

Negli esempi seguenti, l'input e l'output sono distinguibili per la presenza o meno dei prompt ('>>> ' e '... '). Per sperimentare l'esempio è necessario digitare tutto quello che segue il prompt.

Le righe che non iniziano con un prompt sono output dell'interprete.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from interpret.glassbox import ExplainableBoostingClassifier
import matplotlib.pyplot as plt
import numpy as np

# Load a sample dataset
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train an EBM model
ebm = ExplainableBoostingClassifier()
ebm.fit(X_train, y_train)

# Make predictions
y_pred = ebm.predict(X_test)
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")

# Interpret the model
ebm_global = ebm.explain_global(name='EBM')

# Extract feature importances
feature_names = ebm_global.data()['names']
importances = ebm_global.data()['scores']

# Sort features by importance
sorted_idx = np.argsort(importances)
sorted_feature_names = np.array(feature_names)[sorted_idx]
sorted_importances = np.array(importances)[sorted_idx]

# Increase spacing between the feature names
y_positions = np.arange(len(sorted_feature_names)) * 1.5 # Increase multiplier for more space

# Plot feature importances
plt.figure(figsize=(12, 14)) # Increase figure height if necessary
plt.barh(y_positions, sorted_importances, color='skyblue', align='center')
plt.yticks(y_positions, sorted_feature_names)
plt.xlabel('Importance')
plt.title('Feature Importances from Explainable Boosting Classifier')
plt.gca().invert_yaxis()

# Adjust spacing
plt.subplots_adjust(left=0.3, right=0.95, top=0.95, bottom=0.08) # Fine-tune the margins if needed

plt.show()
```

# Commenti in Python

In Python i commenti iniziano con il carattere 'hash', '#' e continuano fino alla fine della riga. Un commento può comparire all'inizio di una riga, dopo degli spazi bianchi o dopo del codice, ma non dentro una stringa costante.

```
# questo è il primo commento  
SPAM = 1 # e questo è il secondo  
STRING = "# Questo non è un commento."
```



# Usare Python come Calcolatrice

L'interprete si comporta come una semplice calcolatrice: si può digitare un'espressione ed esso fornirà il valore risultante. La sintassi delle espressioni è chiara: gli operatori +, -, \* e / funzionano come nella maggior parte degli altri linguaggi.

```
>>> 2+2  
4  
>>> (50-5*6)/4  
5
```

# Numeri e Operazioni Aritmetiche



## Operatori Base

Gli operatori `+, -, * e /` funzionano come atteso. Le parentesi possono essere usate per raggruppare operatori e operandi.



## Divisione Intera

Una divisione tra interi restituisce solo il quoziente: `7/3` restituisce 2, mentre `7/-3` restituisce -3.



## Virgola Mobile

Le operazioni in virgola mobile sono pienamente supportate. In presenza di operandi di tipo misto gli interi vengono convertiti.

**x = 5**



## Assegnamento di Variabili

Il segno di uguale ('=') è usato per assegnare un valore ad una variabile. Il valore di un assegnamento non viene stampato:

```
>>> larghezza = 20
>>> altezza = 5*9
>>> larghezza * altezza
900
```

Un valore può essere assegnato simultaneamente a variabili diverse:

```
>>> x = y = z = 0 # Zero x, y e z
```

# PYTHON

## AI IMMEDDS



## Numeri Complessi

Anche i numeri complessi vengono supportati; per contrassegnare i numeri immaginari si usa il suffisso 'j' o 'J'. I numeri complessi con una componente reale non nulla vengono indicati come '(real+imag j)'.

```
>>> 1j * 1j
(-1+0j)
>>> 3+1j*3
(3+3j)
>>> (1+2j)/(1+1j)
(1.5+0.5j)
```

# Parti Reale e Immaginaria

## Estrazione Componenti

Per estrarre le parti da un numero complesso z si usino z.real e z.imag.

```
>>> a=1.5+0.5j  
>>> a.real  
1.5  
>>> a.imag  
0.5
```

## Conversioni

Le funzioni float(), int() e long() non funzionano con i numeri complessi. Usate abs(z) per ottenere la grandezza.

```
>>> abs(a)  
5.0
```

# Enhance Code Readability with Underscore Placeholders in Python



**Did you know?** Python allows the use of underscores \_ as placeholders in large numbers for better readability! This doesn't affect the value, but makes your code easier to understand at a glance

**Key Takeaway:** Using underscores helps in visualizing large numbers clearly, making your code cleaner and more maintainable



## La Variabile Underscore

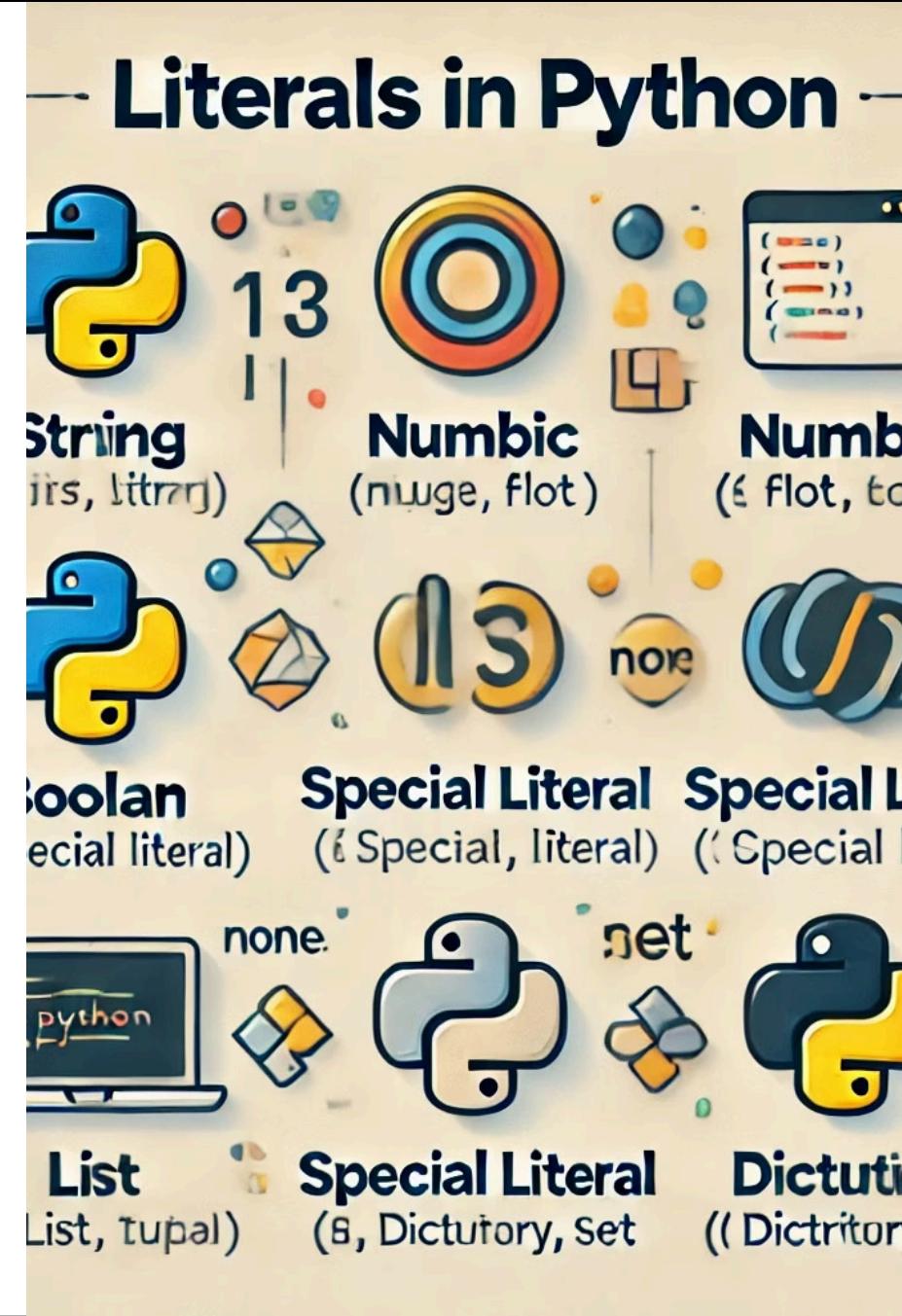
In modo interattivo, l'ultima espressione stampata è assegnata alla variabile '\_'. Questo facilita i calcoli in successione quando si sta usando Python come calcolatrice da tavolo:

```
>>> tassa = 12.5 / 100
>>> prezzo = 100.50
>>> prezzo * tassa
12.5625
>>> prezzo + _
113.0625
>>> round(_, 2)
113.06
```

# Stringhe in Python

Oltre ai numeri, Python può anche manipolare stringhe, che possono essere espresse in diversi modi. Possono essere racchiuse tra apici singoli o virgolette:

```
>>> 'spam eggs'  
'spam eggs'  
>>> "doesn't"  
"doesn't"  
>>> '"Yes," he said.'  
"Yes," he said.'
```



# Stringhe Multiriga

Le stringhe costanti possono estendersi su più righe in modi diversi. Si possono scrivere lunghe righe usando le barre oblique inverse come ultimo carattere di riga:

```
ciao = "Questa è una stringa abbastanza lunga che  
contiene\n\  
parecchie righe di testo proprio come si farebbe in C.\n\  
Si noti che gli spazi bianchi all'inizio della riga sono\  
significativi."
```



## Code:

```
String1 = 'A Beginner\'s'  
String2 = "Guide To"  
String3 = ""Python Strings""  
String4 = String1 + ' ' + String2 + ' ' +  
         print(String4)
```

## Output:

**Beginner's Guide - Python Strings**

# Stringhe Raw e Triple Quotes

## Stringhe Raw

Le stringhe letterali "raw" non convertono le sequenze di escape. Si definiscono con il prefisso 'r'.

## Triple Virgolette

Le stringhe possono essere circondate da triple virgolette ("""" o """"). Non è necessario proteggere i caratteri di fine riga.

# Concatenation

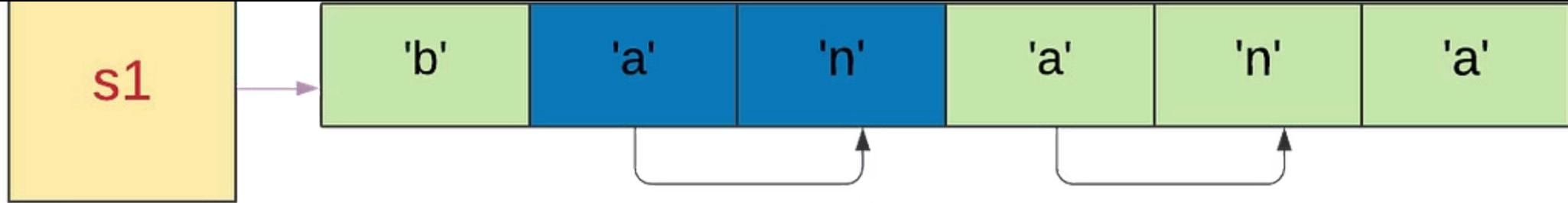
## (Learn With Code Explanations)

### Operazioni sulle Stringhe

Le stringhe possono essere concatenate (incollate assieme) tramite l'operatore + e ripetute tramite \*:

```
>>> parola = 'Aiuto' + 'A'  
>>> parola  
'AiutoA'  
>>> '<' + parola*5 + '>'  
"
```

Due stringhe letterali consecutive vengono concatenate automaticamente.



Determines the length of the string

## Indicizzazione delle Stringhe

Le stringhe possono essere indicizzate come in C, il primo carattere di una stringa ha indice 0. Non c'è alcun tipo associato al carattere; un carattere è semplicemente una stringa di lunghezza uno.

```
>>> parola[4]  
'o'  
>>> parola[0:2]  
'Ai'  
>>> parola[2:4]  
'ut'
```

# Slicing delle Stringhe

Gli indici della fetta hanno utili comportamenti predefiniti. Il primo indice, se omesso, viene impostato a 0. Se viene tralasciato il secondo, viene impostato alla dimensione della stringa:

```
>>> parola[:2] # I primi due caratteri  
'Ai'  
>>> parola[2:] # Tutti eccetto i primi due caratteri  
'utoA'
```

string

s = "immutable"

## Immutabilità delle Stringhe

A differenza di quanto avviene in C, le stringhe Python non possono essere modificate. Un'assegnazione effettuata su un indice di una stringa costituisce un errore.

Tuttavia, creare una nuova stringa combinando i contenuti è semplice ed efficiente:

```
>>> 'x' + parola[1:]  
'xiutoA'  
>>> 'Lavagna' + parola[5]  
'LavagnaA'
```

## String

V	E	G	A	S	T	A	C	K
---	---	---	---	---	---	---	---	---

## Positive Indesing

0	1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---	---

## Negative Indexing

-9	-8	-7	-6	-5	-4	-3	-2	-1
----	----	----	----	----	----	----	----	----

# Indici Negativi

Gli indici possono essere numeri negativi, per iniziare il conteggio da destra:

```
>>> parola[-1] # L'ultimo carattere  
'A'  
>>> parola[-2] # Il penultimo carattere  
'o'  
>>> parola[-2:] # Gli ultimi due caratteri  
'oA'  
>>> parola[:-2] # Tutta la stringa eccetto i due ultimi  
caratteri  
'Aiut'
```

<code>upper()</code>	<code>capitalize()</code>	<code>rindex()</code>	<code>isdigit()</code>	<code>isnumeric()</code>	<code>partition()</code>	<code>format()</code>
<code>lower()</code>	<code>swapcase()</code>	<code>count()</code>	<code>isascii()</code>	<code>isidentifier()</code>	<code>splitlines()</code>	<code>rjust()</code>
<code>title()</code>	<code>replace()</code>	<code>strip()</code>	<code>isspace()</code>	<code>startswith()</code>	<code>removeprefix()</code>	<code>len()</code>

## Visualizzazione degli Indici

Il modo migliore di tenere a mente come lavorano le fette è di pensare che gli indici puntino tra i caratteri, con l'indice 0 posto al margine sinistro del primo carattere:

```
+---+---+---+---+---+
| A | i | u | t | o | A |
+---+---+---+---+---+
 0 1 2 3 4 5 6
 -6 -5 -4 -3 -2 -1
```

La fetta da `i` a `j` consiste di tutti i caratteri compresi tra i margini contrassegnati `i` e `j`.

# find the length of a string

## Lunghezza delle Stringhe

La funzione built-in `len()` restituisce la lunghezza di una stringa:

```
>>> s = 'supercalifragicospiristicalidoso'  
>>> len(s)  
32
```

Le stringhe e le stringhe Unicode sono esempi di tipi sequenza e supportano operazioni comuni consentite su simili tipi.

## Unicode string

```
txt'hhh') as f:
```



# Stringhe Unicode

A partire da Python 2.0 è disponibile un nuovo tipo di dato testo: l'oggetto Unicode. Può essere utilizzato per immagazzinare e manipolare dati Unicode e si integra al meglio con gli oggetti stringa esistenti.

Unicode ha il vantaggio di fornire un unico ordinale per ogni carattere che possa comparire in un qualsiasi testo.

## Raw string

```
txt'hhh') as f:
```



## Unicode string with escape

```
.txt'hhh') as f:
```



# Str vs Unicode

**str: a sequence of bytes**

**unicode: a sequence of code points (ur)**

```
g = "Hello World"  
string)
```

```
le = u"Hi \u2119\u01b4\u2602\u210c\xf8\U0001f60e" #  
unicode)  
le '>
```

## Creazione di Stringhe Unicode

Creare stringhe Unicode in Python è semplice quanto creare stringhe normali:

```
>>> u'Ciao mondo !'  
u'Ciao mondo !'
```

Il carattere 'u' minuscolo davanti agli apici indica che si vuole creare una stringa Unicode. Se si desidera includere caratteri speciali, lo si può fare usando la codifica Python Unicode-Escape:

```
>>> u'Ciao\u0020mondo !'  
u'Ciao mondo !'
```

# Other implicit conversions

```
%s" % my_unicode
\u2119\u01b4\u2602\u210c\xf8\u1f24'

%s" % my_string
'Hello World'

unicode
  (most recent call last):
  Error: 'ascii' codec can't encode character
        at position 3-8: ordinal not in range(128)

encode('utf-8')      # silly
  (most recent call last):
  Error: 'ascii' codec can't decode byte
        at position 3: ordinal not in range(128)

g.encode('utf-8')    # silly
```

# Codifiche Unicode

## Modalità Raw

C'è anche una modalità raw per Unicode. Si deve prefissare 'ur' alla stringa per usare la codifica Raw-Unicode-Escape.

## Conversioni

La funzione built-in `unicode()` permette l'accesso a tutti i codec Unicode ufficiali per convertire tra diverse codifiche.

## Data Structures

itive

Float

Boolean

Non-Primit

Built-In

List

Tuple

Set

Dictionary

# Liste in Python

Python riconosce una certa quantità di tipi di dati composti, usati per raggruppare insieme altri valori. Il più versatile è il tipo lista, che può essere scritto come una lista di valori separati da virgole tra parentesi quadre.

```
>>> a = ['spam', 'eggs', 100, 1234]  
>>> a  
['spam', 'eggs', 100, 1234]
```

Gli elementi della lista non devono essere necessariamente tutti dello stesso tipo.



# Operations

## Python

by Hanzel Godinez

@GodinezHanzel

:

Creates an empty list

Adds a single element to the end  
of a list

Adds another list to the end of the

Inserts a new element at a given  
position in the list

Removes a list element or slice

Searches for and removes a given  
value from a list

Inverses a list in place

Sorts a list in place

Immutable operations	
sorted	Sorts a list without changing the original list
+	Adds two lists
*	Replicates a list
min	Returns the minimum value in a list
max	Returns the maximum value in a list
index	Returns the index of the first occurrence of a value in a list
count	Counts the number of times a value occurs in a list
sum	Sum the items in an iterable (if all items are summed)
in	Returns whether an element is present in a list

# Operazioni sulle Liste

Come per gli indici delle stringhe, gli indici delle liste iniziano da 0, e anche le liste possono essere affettate, concatenate e così via:

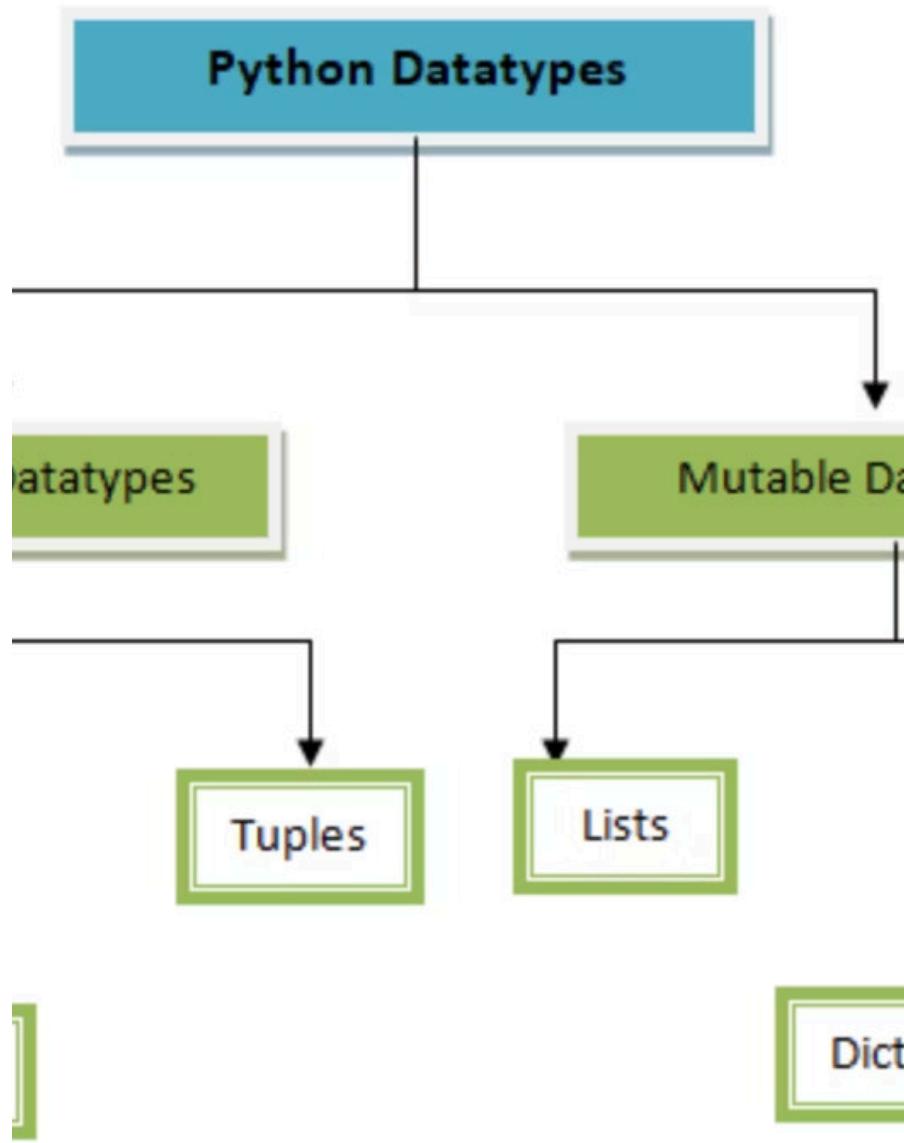
```
>>> a[0]  
'spam'  
>>> a[-2]  
100  
>>> a[1:-1]  
['eggs', 100]  
>>> a[:2] + ['bacon', 2*2]  
['spam', 'eggs', 'bacon', 4]
```

# Mutabilità delle Liste

Al contrario delle stringhe, che sono immutabili, è possibile modificare gli elementi individuali di una lista:

```
>>> a = ['spam', 'eggs', 100, 1234]  
>>> a[2] = a[2] + 23  
>>> a  
['spam', 'eggs', 123, 1234]
```

È anche possibile assegnare valori alle fette, e questo può pure modificare le dimensioni della lista.



# Assegnamenti alle Fette

01

## Rimpiazzare Elementi

`a[0:2] = [1, 12]` sostituisce i primi due elementi

02

## Rimuovere Elementi

`a[0:2] = []` rimuove i primi due elementi

03

## Inserire Elementi

`a[1:1] = ['bletch', 'xyzzy']` inserisce elementi nella posizione 1

04

## Inserire All'Inizio

`a[:0] = a` inserisce una copia di se stesso all'inizio



# Nested Lists in Guide

## Liste Annidate

È possibile avere delle liste annidate (contenenti cioè altre liste):

```
>>> q = [2, 3]
>>> p = [1, q, 4]
>>> len(p)
3
>>> p[1]
[2, 3]
>>> p[1][0]
2
>>> p[1].append('xtra')
>>> p
[1, [2, 3, 'xtra'], 4]
```

Si noti che p[1] e q si riferiscono proprio allo stesso oggetto!

# Primi Passi verso la Programmazione

Possiamo usare Python per compiti più complessi. Per esempio, possiamo scrivere una sottosuccessione iniziale della serie di Fibonacci:

```
>>> # La serie di Fibonacci:  
... # la somma di due elementi definisce l'elemento successivo  
... a, b = 0, 1  
>>> while b < 10:  
...     print b  
...     a, b = b, a+b  
...  
1  
1  
2  
3  
5  
8
```



# Caratteristiche dell'Esempio Fibonacci

## 1 Assegnamento Multiplo

La prima riga contiene un assegnamento multiplo: le variabili a e b ricevono simultaneamente i nuovi valori 0 e 1.

## 2 Ciclo While

Il ciclo while viene eseguito fino a quando la condizione ( $b < 10$ ) rimane vera. In Python, qualsiasi valore diverso da zero è vero.

## 3 Indentazione

Il corpo del ciclo è indentato: l'indentazione è il sistema con cui Python raggruppa le istruzioni.

## 4 Istruzione Print

L'istruzione print stampa il valore dell'espressione, gestendo espressioni multiple e stringhe in modo elegante.

# Operatori di Confronto

Gli operatori standard di confronto sono scritti come in C: < (minore di), > (maggiore di), == (uguale a), <= (minore o uguale a), >= (maggiore o uguale a) e != (diverso da).

La condizione può anche essere il valore di una stringa o di una lista; qualsiasi cosa con una lunghezza diversa da zero ha valore logico vero, sequenze vuote hanno valore logico falso.

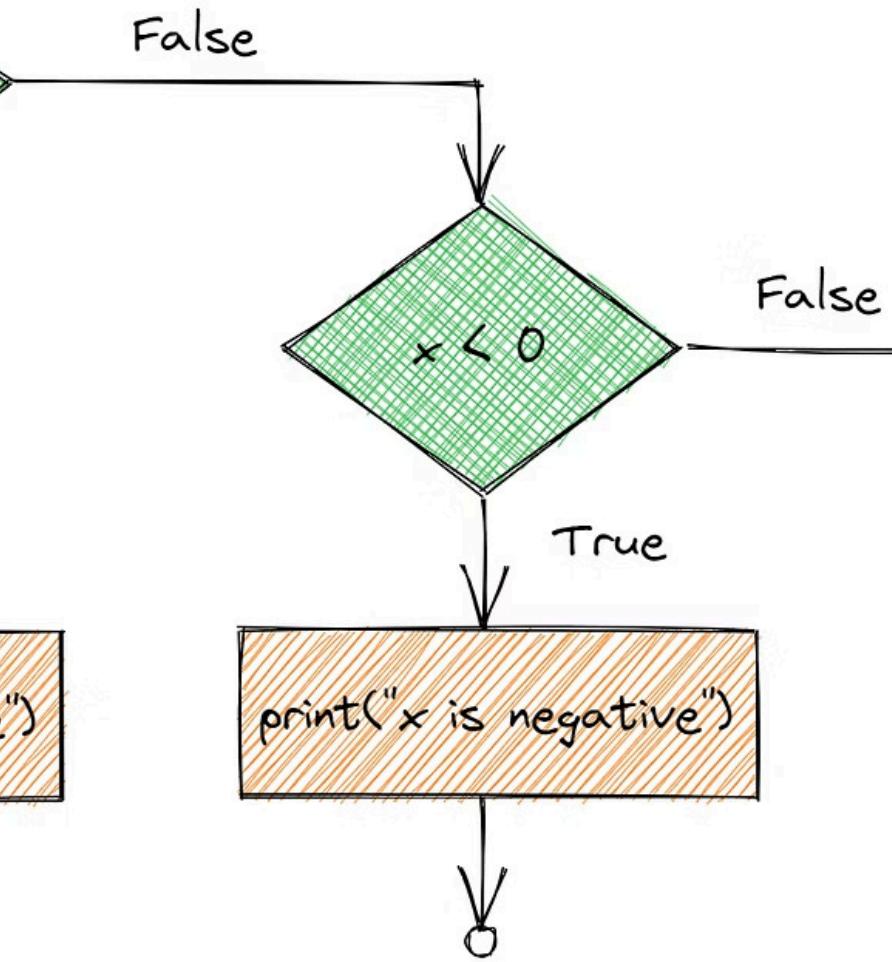
## What Are Logical Operators In Python

### Description

**Returns True if both operands are True**

**turns True if either of the operands are True**

**Retruns True if the operand in False**



# Strumenti di Controllo del Flusso

Oltre all'istruzione while, Python riconosce le solite istruzioni di controllo del flusso presenti in altri linguaggi, con qualche particolarità.

# if-else Statement

The simplest form of conditional logic in Python is the if-else statement. It checks whether a condition is true or false and executes a specific block of code based on that.

```
10  
x > 5:  
    print("x is greater than 5")  
else:  
    print("x is less than or equal to 5")  
  
greater than 5
```

If `x` is greater than 5, the first block will run; otherwise, the second block runs.

## L'Istruzione if

Il tipo di istruzione più conosciuta è if. Possono essere presenti una o più parti elif, e la parte else è facoltativa:

```
>>> x = int(raw_input("Introdurre un numero: "))  
>>> if x < 0:  
...     x = 0  
...     print 'Numero negativo cambiato in zero'  
... elif x == 0:  
...     print 'Zero'  
... elif x == 1:  
...     print 'Uno'  
... else:  
...     print 'Più di uno'
```

La parola chiave 'elif' è un'abbreviazione di 'else if' e serve ad evitare un eccesso di indentazioni.

Initialization

Stop a loop

1 for i in range(1,11):

Block of

## L'Istruzione for

L'istruzione for di Python differisce da quella di C o Pascal. Piuttosto che iterare sempre su una progressione aritmetica, in Python l'istruzione for compie un'iterazione sugli elementi di una qualsiasi sequenza:

```
>>> # Misura la lunghezza di alcune stringhe:  
... a = ['gatto', 'finestra', 'defenestrare']  
>>> for x in a:  
... print x, len(x)  
...  
gatto 5  
finestra 8  
defenestrare 12
```

# iterate over a list

## Modificare Sequenze Durante l'Iterazione

Non è prudente modificare all'interno del ciclo la sequenza su cui avviene l'iterazione. Se è necessario modificare la lista su cui si effettua l'iterazione, si deve iterare su una copia:

```
>>> for x in a[:]: # fa una copia tramite affettamento dell'intera lista
...     if len(x) > 6: a.insert(0, x)
...
>>> a
['defenestrare', 'gatto', 'finestra', 'defenestrare']
```

La notazione a fette rende questo procedimento particolarmente conveniente.

# La Funzione range()

Se è necessario iterare su una successione di numeri, viene in aiuto la funzione built-in range(), che genera liste contenenti progressioni aritmetiche:

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(5, 10)
[5, 6, 7, 8, 9]
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> range(-10, -100, -30)
[-10, -40, -70]
```

L'estremo destro passato alla funzione non fa mai parte della lista generata.



# Iterazione sugli Indici

Per effettuare un'iterazione sugli indici di una sequenza, si usino in combinazione range() e len():

```
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
>>> for i in range(len(a)):
...     print i, a[i]
...
0 Mary
1 had
2 a
3 little
4 lamb
```

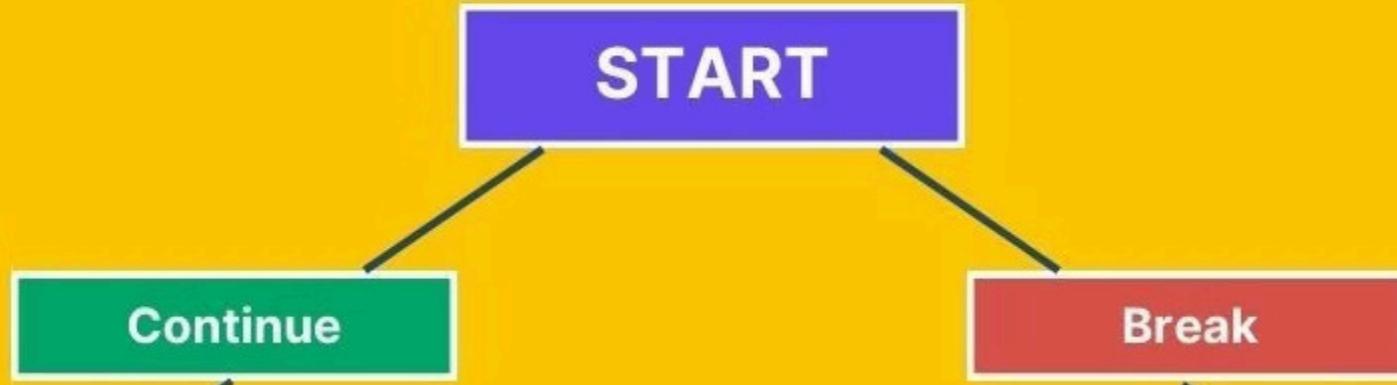
## range() vs indexing

```
range(len(my_list)):
    list[i]      # indexing!
```

ul:

```
enumerate(iterable):
    v
```

```
, line in enumerate(f, start=1):
```



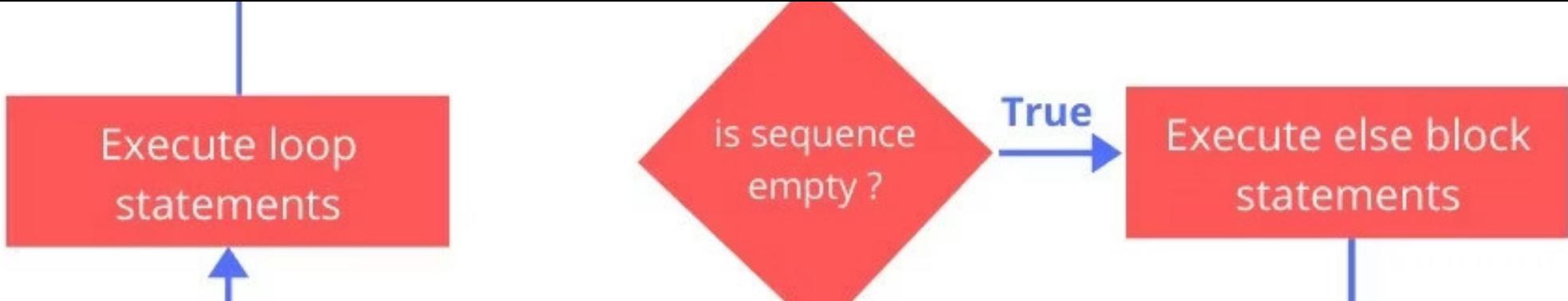
## Le Istruzioni break e continue

### Break

L'istruzione break, come in C, esce immediatamente dal ciclo for o while più interno che la racchiude.

### Continue

L'istruzione continue, anch'essa presa a prestito dal C, prosegue con l'iterazione seguente del ciclo.



## La Clausola else nei Cicli

Le istruzioni di ciclo possono avere una clausola else che viene eseguita quando il ciclo termina per esaurimento della lista (con for) o quando la condizione diviene falsa (con while), ma non quando il ciclo è terminato da un'istruzione break.

```
>>> for n in range(2, 10):
... for x in range(2, n):
... if n % x == 0:
... print n, 'è uguale a', x, '*', n/x
... break
... else:
... print n, 'è un numero primo'
```

# Pass Statement In

```
class Student:  
    print('Inside Student class')  
    pass  
    print('Pass executed')
```



It is a placeholder that does nothing when executed

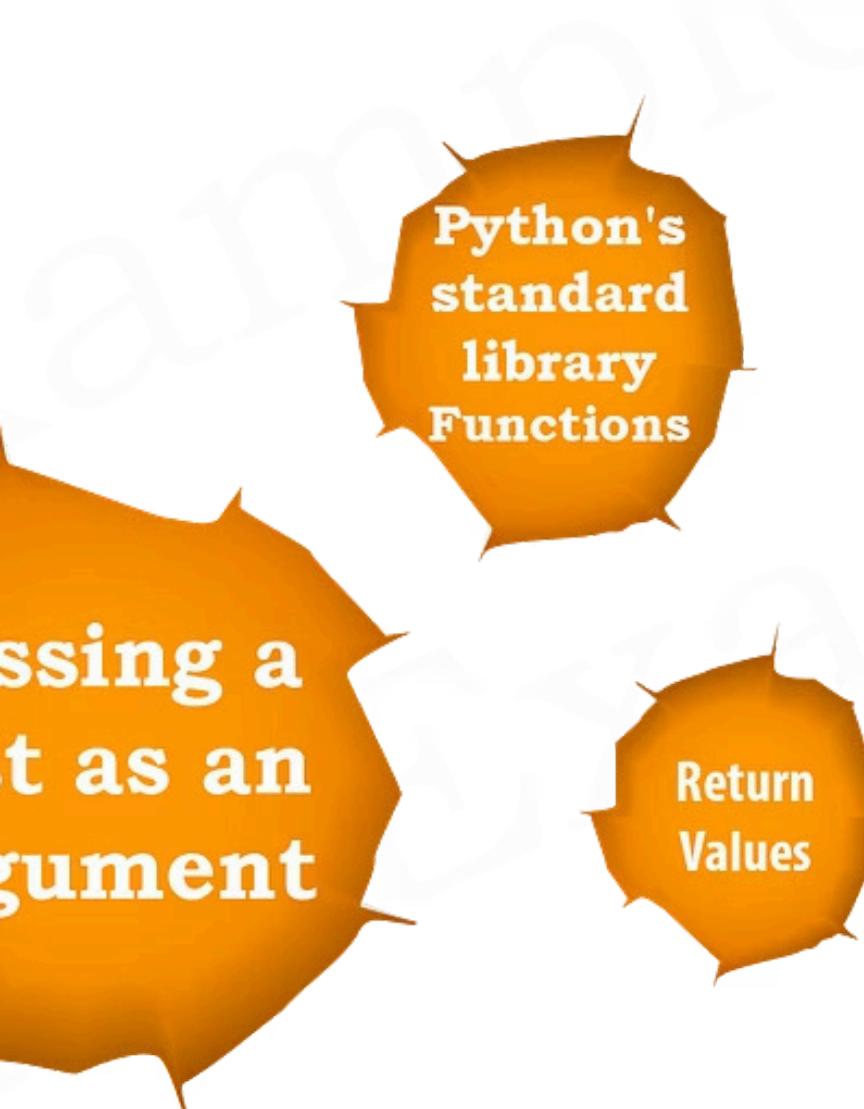
## L'Istruzione pass

L'istruzione pass non fa nulla. Può essere usata quando un'istruzione è necessaria per sintassi ma il programma non richiede venga svolta alcuna azione:

```
>>> while True:  
...     pass # In attesa di un interrupt da tastiera
```

È utile come placeholder durante lo sviluppo del codice.

# n Functions -



## Definizione di Funzioni

Possiamo creare una funzione che scrive la serie di Fibonacci fino ad un limite arbitrario:

```
>>> def fib(n): # scrive la serie di Fibonacci fino a n  
... "Stampa una serie di Fibonacci fino a n"  
... a, b = 0, 1  
... while b < n:  
...     print b,  
...     a, b = b, a+b  
...  
>>> fib(2000)  
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

# Caratteristiche delle Funzioni

## Parola Chiave def

La parola chiave def introduce una definizione di funzione, seguita dal nome della funzione e dalla lista dei parametri formali.

## Docstring

La prima istruzione del corpo della funzione può essere facoltativamente una stringa di documentazione o docstring.

## Spazio dei Nomi

L'esecuzione di una funzione introduce una nuova tabella di simboli, usata per le variabili locali della funzione.

Param

# Passaggio di Parametri

I parametri attuali di una chiamata a funzione vengono introdotti nella tabella dei simboli locale della funzione al momento della chiamata. Gli argomenti sono passati usando una chiamata per valore, dove il valore è sempre un riferimento ad un oggetto.

Una definizione di funzione introduce il nome della funzione nella tabella dei simboli corrente. Il valore può essere assegnato a un altro nome che quindi può venire anch'esso usato come una funzione.

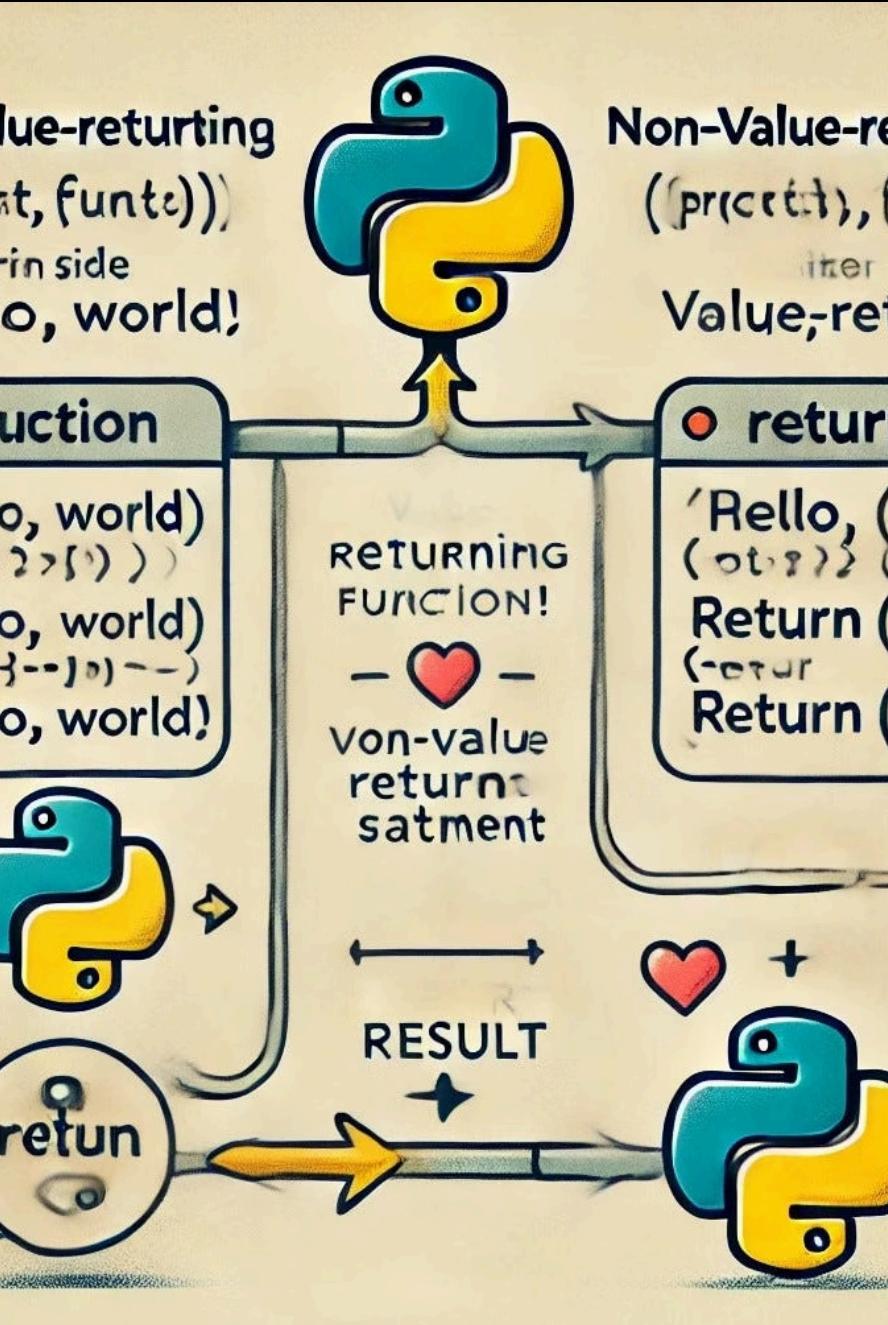
ion Definition

**add(a, b):**  
**eturn a + b**

n Call

**|(2, 3)**

Argument



## Funzioni che Restituiscono Valori

È facile scrivere una funzione che restituisce una lista dei numeri delle serie di Fibonacci anziché stamparli:

```
>>> def fib2(n): # restituisce la serie di Fibonacci fino a n
...     """Restituisce una lista contenente la serie di Fibonacci fino a n"""
...     result = []
...     a, b = 0, 1
...     while b < n:
...         result.append(b) # aggiunge b alla lista
...         a, b = b, a+b
...     return result
...
>>> f100 = fib2(100)
>>> f100
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

L'istruzione `return` restituisce una funzione con un valore. Il metodo `append()` aggiunge un nuovo elemento alla fine della lista.