

Programmazione in Python

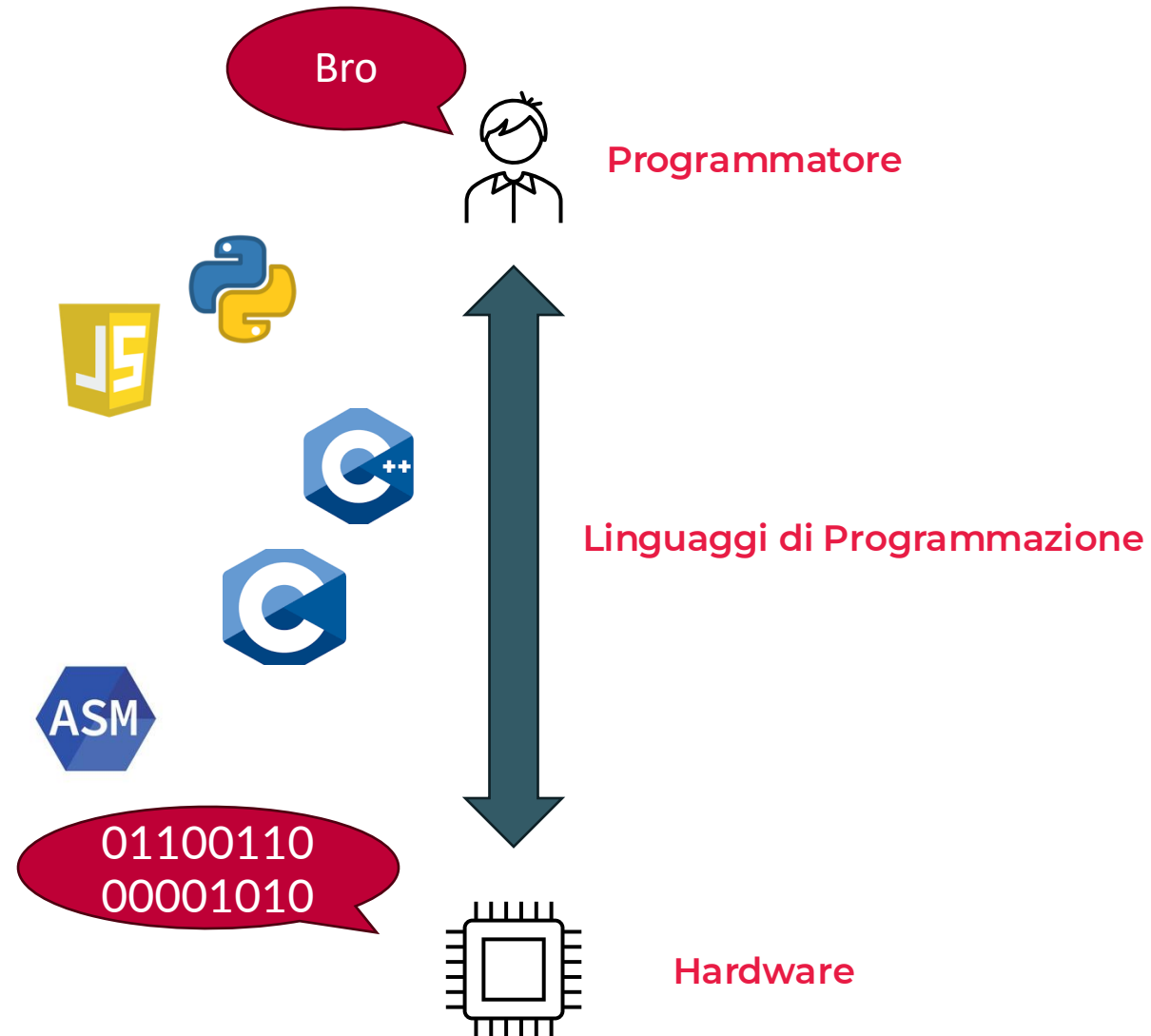
09/2025

Docenti: Michele Crudele, Francesca Laera

Sommario

- Introduzione
- Forme di Python
- Installazione
- Operatori
- Variabili
- Condizioni
- Loops

Cos'è Python?



The Python Language


Python è un linguaggio di programmazione di **alto livello e general-purpose**. La sua filosofia di progettazione enfatizza la leggibilità del codice con l'uso di indentazioni.

È stato creato nel 1989 da Guido Van Rossum come progetto amatoriale per tenersi impegnato durante le vacanze di Natale. Attualmente è alla versione **3.13**

Oggi, grazie all'elevato numero di librerie esterne, il suo utilizzo si è diffuso in diversi ambiti: sviluppo web, sviluppo di interfacce grafiche utente (GUI), applicazioni scientifiche, ecc.

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
```

 Use Python for... [>>> More](#)

Web Development: Django , Pyramid , Bottle , Tornado , Flask , web2py

GUI Development: tkinter , PyGObject , PyQt , PySide , Kivy , wxPython , DearPyGui

Scientific and Numeric: SciPy , Pandas , IPython

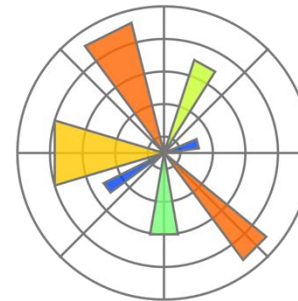
Software Development: Buildbot , Trac , Roundup

System Administration: Ansible , Salt , OpenStack , xonsh

Phyton in Data Science

In particolare, Python è diventato lo standard di fatto per la **Data Science**.

- Librerie con ampie raccolte di funzioni matematiche e strumenti per i dati: Pandas, NumPy, ...
- Librerie per la visualizzazione dei dati: Matplotlib, Seaborn, Dash, ...
- Librerie per Machine Learning e Deep Learning: Scikit-learn, Tensorflow, Torch, ...



Perché Python?

Perché è facile!

1. Facile da imparare
Lo vedrete
2. Facile da interfacciare
Con ogni genere di cosa esistente
3. Facile da installare
Come vedremo a breve
4. Community gigantesca e attiva



Cosa si può fare con Python?

- Sviluppo di siti Web. Instagram, Google, Netflix e Uber sono esempi di siti web che usano python come backend;
- Videogames : Battlefield 2 usa python per alcune funzionalità, EVE online e Mount & Blade sono scritti interamente in python;
- Calcolo scientifico, esperimenti e Intelligenza Artificiale. Praticamente tutta l'IA che pervade la nostra epoca è scritta in python.



Perché non Python?

1. Non ha il massimo delle performance
2. Non è adatto a programmazione di basso livello (con alcune eccezioni)
3. Non è adatto a programmare app mobili (non ci sono molti pacchetti con performance competitive su quel fronte)



Installare Python

Installate l'ultima versione di Python dal sito ufficiale.

Ricordate di settare le variabili di ambiente necessarie.

Provare ad avviare python da terminale con *python -V*



Notebooks

Un notebook è uno strumento interattivo utilizzato per scrivere ed eseguire codice, visualizzare risultati e documentare il lavoro in modo integrato.

- **Programmazione interattiva:** permette di eseguire codice in piccoli blocchi chiamati "celle".
- **Output immediato:** mostra i risultati subito dopo l'esecuzione del codice.
- **Integrazione di testo e media:** consente di aggiungere spiegazioni, immagini e grafici accanto al codice.
- **Facilità di condivisione:** ideale per collaborare o condividere progetti con altri.
- **Usi comuni:** analisi dati, apprendimento automatico, educazione e ricerca.

Esempi popolari di notebook includono Jupyter Notebook e Google Colab.



Notebooks

1. Create un file <nome_a_piacere>.ipynb
2. Apritelo in Visual Studio Code
3. Create una cella di markdown (testo) e scrivete un commento acido
4. Create una cella di code (codice) e scrivete un semplice print



La console python

Una volta installato, è possibile richiamare la console di python semplicemente scrivendo:

```
python
```

all'interno di una shell, verrà visualizzato il prompt:

```
>>
```

Qui si possono dare istruzioni python una riga alla volta:

```
>> print("Ciao a tutti!")
```

La funzione `print` consente di scrivere qualcosa a schermo, rendendola utile per mandare messaggi all'utente. Diamo una piccola occhiata a quello che ci aspetta analizzandola più da vicino:

- `print` è una funzione e le parentesi indicano l'input
- `""` servono a indicare il fatto che stiamo trattando una stringa di testo tal quale e non di codice

Script python: i file .py



Ma questo significa che
ogni volta che devo fare
qualsiasi cosa devo
scrivere riga per riga?


No, fortunatamente no, in particolare, possiamo scrivere un file con estensione .py e poi farlo girare usando:

- L'infrastruttura di un IDE
- Il comando `python nome_file.py`



Jupyter Notebooks

Un jupyter notebook è un'infrastruttura che consente di alternare testo e codice, potenzialmente eseguendo il codice una riga per volta e mantenendo le variabili in memoria.



Ma in questo modo la mia RAM sarà intasata in poco tempo!

Il codice in un jupyter notebook viene eseguito all'interno di un terminale virtuale chiamato kernel. quando la RAM viene riempita, il kernel si spegne automaticamente. Questo è uno scenario da evitare, ma salvaguarda il computer da bloccaggi indesiderati. Inoltre, l'uso di un kernel consente di resettare tutte le variabili e tutto quello che si è fatto semplicemente riavviando il kernel





Andiamo sul Codice, no?

Number Types

In Python esistono **tre tipi di numeri**:

- `int()` per i numeri interi (e.g., -3, 0, 42).

Usati per conteggi, indici, quantità discrete.

- `float()` per i numeri in virgola mobile (reali) (e.g., 3.14, -0.001, 2.7e5).

Usati per misurazioni precise, calcoli scientifici, valori continui.

- `complex()` per i numeri complessi (parte reale + parte immaginaria)

(e.g., 3+5j, `complex(2, -3)`). Usati in specifici domini scientifici (es. fisica quantistica).



```
print(5)
print("as integer: ", int(5))
print("as floating-point: ", float(5))
print("as complex: ", complex(5))
```

```
""" 5
>>> as integer: 5
>>> as floating-point: 5.0
>>> as complex: (5+0j)
```



Arithmetic Operations

A livello fondamentale, Python può essere utilizzato come una grande calcolatrice

Operation	Result
<code>x + y</code>	sum of <code>x</code> and <code>y</code>
<code>x - y</code>	difference of <code>x</code> and <code>y</code>
<code>x * y</code>	product of <code>x</code> and <code>y</code>
<code>x / y</code>	quotient of <code>x</code> and <code>y</code>
<code>x // y</code>	floored quotient of <code>x</code> and <code>y</code>
<code>x % y</code>	remainder of <code>x / y</code>
<code>-x</code>	<code>x</code> negated
<code>+x</code>	<code>x</code> unchanged

<code>abs(x)</code>	absolute value or magnitude of <code>x</code>
<code>int(x)</code>	<code>x</code> converted to integer
<code>float(x)</code>	<code>x</code> converted to floating point
<code>complex(re, im)</code>	a complex number with real part <code>re</code> , imaginary part <code>im</code> . <code>im</code> defaults to zero.
<code>c.conjugate()</code>	conjugate of the complex number <code>c</code>
<code>divmod(x, y)</code>	the pair <code>(x // y, x % y)</code>
<code>pow(x, y)</code>	<code>x</code> to the power <code>y</code>
<code>x ** y</code>	<code>x</code> to the power <code>y</code>



Arithmetic Operations

Operator	Precedence
<code>**</code>	highest precedence
<code>*</code> , <code>/</code> , <code>//</code> , <code>%</code>	
<code>+</code> , <code>-</code>	lowest precedence

```
risultato = 10 + 5 * 2 - 16 / 2 ** 2
print(risultato) # Output: 16.0
# Passaggi di valutazione:
# 1. Potenza: 2 ** 2 => 4
# 2. Divisione: 16 / 4 => 4.0
# 3. Moltiplicazione: 5 * 2 => 10
# 4. Addizione: 10 + 10 => 20
# 5. Sottrazione: 20 - 4.0 => 16.0
```

[Built-in Types — Python 3.12.2 documentation](#)



math Module

Il modulo **math** è una **libreria Python integrata** progettata per semplificare le attività matematiche. Fornisce accesso a varie costanti e funzioni matematiche. Per operazioni matematiche di base su singoli numeri, il modulo **math** è perfetto. Per calcoli più complessi su collezioni di dati (come vettori o matrici), librerie più avanzate come NumPy (che vedremo) sono lo standard.

```
import math

math.sqrt(4) # square root
math.exp(2.9) # exponentiation
math.cos(0) # cosine

math.pi # pi
math.e # euler constant

radius = 5
area = math.pi * radius**2
print(f"Area of a circle with radius {radius}: {area}")
```

[math — Mathematical functions — Python 3.12.2 documentation](#)



Esempi

```
import math

print(f"Logaritmo base 10 di 100: {math.log10(100)}")
print(f"Seno di 90 gradi (pi/2 radianti): {math.sin(math.pi/2)}")
print(f"Fattoriale di 5: {math.factorial(5)}")
print(f"Arrotondamento per eccesso di 2.3: {math.ceil(2.3)}")
print(f"Arrotondamento per difetto di 2.7: {math.floor(2.7)}")
```



Variables

Le variabili sono il modo per memorizzare oggetti nella memoria e fare riferimento ad essi con alcuni nomi

- Per creare una variabile, utilizzare l'operatore di assegnazione =
- Python è a tipizzazione dinamica, quindi non è necessario dichiarare esplicitamente il tipo di variabile.
- **Regole:**
 - Non può iniziare con un numero.
 - Sono consentite solo lettere, numeri e caratteri di sottolineatura.
 - Parole Chiave (Keywords): Python ha un insieme di parole riservate (keywords) che hanno un significato speciale e non possono essere usate come nomi di variabili (es: if, for, while, def, class, True, False, None). Potete visualizzarle con `import keyword; print(keyword.kwlist)`.
- **Naming conventions (PEP 8):**
 - Utilizzare la convenzione `snake_case`.
 - La variabile privata inizia con un carattere di underscore.
 - Essere espliciti nel nome della variabile (evitare variabili con un solo carattere).



```
animal = 'panda' # string variable
number = 5.0     # float variable
count = 2        # integer variable

print(animal)
print(count + 4)
print(number * 5)

""" panda
>>> 6
>>> 25.0
```

[PEP 8 – Style Guide for Python Code | peps.python.org](https://peps.python.org/pep-0008/)



Esempio: assegnazione dinamica

```
mia_variabile = 100
print(f"Valore: {mia_variabile}, Tipo: {type(mia_variabile)}")
# Output: Valore: 100, Tipo: <class 'int'>

mia_variabile = "Ciao Python!"
print(f"Valore: {mia_variabile}, Tipo: {type(mia_variabile)}")
# Output: Valore: Ciao Python!, Tipo: <class 'str'>

mia_variabile = 3.14159
print(f"Valore: {mia_variabile}, Tipo: {type(mia_variabile)}")
# Output: Valore: 3.14159, Tipo: <class 'float'>
```

Comparison and Logic

Python valuta l'espressione logica da sinistra a destra. Il risultato è un valore di tipo *booleano* (*True* o *False*).

Comparison operators

- Equality ==
- Inequality !=
- Greater than > and >=
- Less than < and <=

Logical operators

- and
- or
- not

```
# comparison
x = 5
y = 7
result = x == y # False

age = 18
is_adult = age != 18 # False

# logical
has_umbrella = False
is_raining = True
need_umbrella = has_umbrella or is_raining # True

is_weekend = False
is_weekday = not is_weekend # True
```



Esempi

```
# Esempi con operatori logici
eta = 25
possiede_patente = True
puo_guidare = eta >= 18 and possiede_patente # True
print(f"Può guidare: {puo_guidare}")

ha_caffe = False
ha_te = True
puo_bere_qualcosa_di_caldo = ha_caffe or ha_te # True
print(f"Può bere qualcosa di caldo: {puo_bere_qualcosa_di_caldo}")

piove = True
non_piove = not piove # False
print(f"Non piove: {non_piove}")
```



Exercise

1. Il Sistema Geodetico Mondiale (WGS-84) è un insieme di standard internazionali per la descrizione della forma della Terra. Nell'ultima revisione del WGS-84, la terra è approssimata ad un ellissoide con semiasse maggiore e semiasse minore $a = 6\,378\,127,0$ m e $b = 6\,356\,752,314245$ m.

Utilizzare la formula per l'area superficiale di un ellissoide.

$$S = 2\pi a^2 \left(1 + \frac{1 - e^2}{e} \operatorname{atanh} e \right)$$

dove $e^2 = 1 - \frac{b^2}{a^2}$, per calcolare l'area superficiale della Terra.

RISULTATO $S = 509\,991\,631 \text{ km}^2$



Esercizi

Calcola il prezzo finale di un articolo dopo aver applicato uno sconto basato sull'importo speso.

Regole Sconto (da implementare con variabili e confronti booleani)

- Spesa inferiore a 50€ : nessuno sconto
- Spesa da 50€ a 99.99€: sconto del 5%
- Spesa da 100€: sconto del 10%

Strings

Un oggetto stringa Python è una sequenza **ordinata** e **immutabile** di caratteri.

- **ordinata** : una stringa è un array di caratteri indicizzati (primo indice 0).
- **immutabile** non può essere modificata tramite assegnazione.

```
# string are defined through single quotes
a = 'cat'
# or through double quotes ""
a = "cat"
print(a)
>>> cat

# character can be accessed by indexing

print(a[0])
print(a[1])
print(a[2])

>>> c
>>> a
>>> t
```



Strings basics

- Le stringhe sono accessibili tramite un indice. Sono validi anche gli indici negativi.
- L'accesso a un indice al di fuori della lunghezza della stringa genera un errore.
- Anche il tipo numerico ha una rappresentazione in forma di stringa.

P Y T H O N

1 2 3 4 5 6

-6 -5 -4 -3 -2 -1

```
a = 'cat'

# Accessing to a negative index goes
# backward at the end of the string
print(a[-1])
>>> t

# Accessing an index outside the string length
# gives an error
a[4]
>>> Traceback (most recent call last):
>>> File "<stdin >", line 1, in <module >
>>> IndexError: string index out of range

# a number can be converted in string in this way
n = str(5.0)
print(n)
>>> 5.0
```



String operations

Sono supportate le operazioni aritmetiche più comuni:

- L'operatore di addizione `+` serve per concatenare stringhe
- L'operatore di moltiplicazione `*` viene utilizzato per ripetere uno o più caratteri
- NB: queste operazioni funzionano solo tra stringhe!!

ESERCIZIO: Stampare un'intestazione utilizzando operazioni di stringa

```
-----
+           HEADER           +
-----
```

```
print('abc' + 'defg')
print('abc' * 4)
print( ('-' * 4 + '=' * 2) * 10 )

>>> abcdefg
>>> abcabcabcab
>>> -----==-----==
```

```
eta = 25
# messaggio = "Ho " + eta + " anni." # TypeError!
messaggio_corretto = "Ho " + str(eta) + " anni."
print(messaggio_corretto)
```



Esercizio svolto

```
linea_trattini = '-' * 40
testo_header = "HEADER"
spazio_laterale = (40 - len(testo_header) - 2) // 2 # -2 per i '+'
spazio_extra = (40 - len(testo_header) - 2) % 2 # per centratura dispari
linea_testo = '+' + ' ' * spazio_laterale + testo_header + ' ' * (spazio_laterale + spazio_extra) + '+'

print(linea_trattini)
print(linea_testo)
print(linea_trattini)
```

Strings formatting

Le variabili possono essere inserite nelle stringhe utilizzando la formattazione delle stringhe

- `format(vars):`
just substitute `{}` in string with vars
- f-strings:
`f"string {variable}"`
- Le cifre decimali e il formato delle variabili sono specificati tramite *specificatori di formato*

```
# string can be formatted through the format method

'{} plus {} equals {}'.format(2, 3, 'five')

'{1} plus {0} equals {2}'.format(2, 3, 'five')

'{num1} plus {num2} equals {answer}'.format(num1=2,
                                             num2=3, answer='five')

# format can also be used to format numbers
# using a specifier

a = 512

print('a = {:5d}'.format(a))    # decimal
print('a = {0:10b}'.format(a))  # binary
print('a = {0:5o}'.format(a))   # octal
print('a = {0:5x}'.format(a))   # hex
```



String Methods

[Built-in Types — Python 3.12.2 documentation](#)

Method	Description
<code>center(width)</code>	Restituisce la stringa centrata in una stringa con larghezza del numero totale di caratteri
<code>endswith(suffix)</code>	Restituisce True se la stringa termina con il suffisso della sottostringa
<code>startswith(prefix)</code>	Restituisce True se la stringa inizia con il prefisso della sottostringa.
<code>index(substring)</code> <code>find(substring)</code>	Restituisce l'indice più basso nella stringa contenente la sottostringa Come sopra. Se la sottostringa non viene trovata, restituisce -1
<code>strip([chars])</code>	Restituisce una copia della stringa con i caratteri iniziali e finali specificati da [chars] rimossi. Se [chars] viene omissso, tutti gli spazi vuoti iniziali e finali vengono rimossi.
<code>upper()</code> / <code>lower()</code>	Restituisce una copia della stringa con tutti i caratteri in maiuscolo/minuscolo.
<code>title()</code>	Restituisce una copia della stringa con tutte le parole che iniziano con la lettera maiuscola e gli altri caratteri in minuscolo.
<code>replace(old, new)</code>	Restituisce una copia della stringa con tutte le parole che iniziano con la lettera maiuscola e gli altri caratteri in minuscolo.
<code>split([sep])</code>	Restituisce un elenco (vedere più avanti) di sottostringhe della stringa originale, separate dalla stringa sep. Se sep non è specificato, il separatore viene considerato un qualsiasi numero di spazi vuoti.
<code>join([list])</code>	Utilizzare la stringa come separatore per unire un elenco di stringhe.
<code>isalpha()</code> / <code>isdigit()</code>	Restituisce True se tutti i caratteri nella stringa sono alfabetici/cifre e la stringa non è vuota; in caso contrario, restituisce False.

String Methods – Some Examples

- `len()` è una funzione integrata che restituisce la lunghezza di un array.
- La presenza di una sottostringa all'interno di una stringa viene verificata con l'operatore `in`.
- Una sottostringa da una stringa può essere estratta tramite slicing
- `s[indice_iniziale:indice_finale]`

EXERCISE: Using slicing, divide the string
'whitechocolatespaceegg'

```
a = 'my python teacher is a very funny person!'

# len built-in function return the length of the array
# (in this case the length of the string)
len(a)

# some useful method to manipulate strings
a.isalpha()
b = a.title()
c = b.replace('funny', 'boring')
c.index('Python')

# the presence of a substring is checked with in
print('python' in c)

# strings can be sliced
c[6:].startswith('Py')
```



Esempi

```
frase = "    Ciao Mondo! Benvenuto in Python.    "  
frase_pulita_e_maiuscola = frase.strip().upper()  
print(frase_pulita_e_maiuscola) # "CIAO MONDO! BENVENUTO IN PYTHON."
```

```
elementi = "mela,banana,ciliegia"  
lista_frutti = elementi.split(',')  
print(lista_frutti) # ['mela', 'banana', 'ciliegia']  
  
nuova_stringa = " - ".join(lista_frutti)  
print(nuova_stringa) # "mela - banana - ciliegia"
```



Esercizi sulle stringhe

1. Si considerino le seguenti sequenze nucleotidiche di coppie di basi (ovvero contenenti solo le lettere A, G, C, T):
TGGATCCA

Data la stringa della sequenza precedente, `sequenza = 'TGGATCCA'`

- Determinare la frazione di basi G e C nella sequenza
HINT: (Usare il metodo `count()` delle stringhe e `len()`)
- Escogitare un modo per determinare se una sequenza nucleotidica è palindroma, nel senso che la sua sequenza complementare è la stessa della sequenza originale letta al contrario. Le coppie di basi complementari sono (A, T) e (C, G) (Suggerimento: dare un'occhiata ai metodi stringa)
Regole: $A \leftrightarrow T$; $C \leftrightarrow G$

HINT: Invertire la sequenza originale, applicare le regole di transizione con il metodo `replace`

Lists

A list is an ordered, mutable array of objects.



```
# list are defined through square brackets []
list1 = [1, 'two', 3.14, 0]

# any object can be an element of a list
a = 34
list2 = [1, a, list1, True]

# an item from the list can be retrieved by
indexing it
print(list[2])
>>> 34

print(list[-1])
True
```

Lists

Una lista è un array ordinato e modificabile di oggetti.

- È possibile accedere alla lista tramite indicizzazione (anche con indici negativi).
- List supporta lo *slicing*

```
# list are defined through square brackets []
list1 = [1, 'two', 3.14, 0]

# any object can be an element of a list
a = 34
list2 = [1, a, list1, True]

# an item from the list can be retrieved by indexing it
print(list[2])
>>> 34

print(list[-1])
True
```



Lists

L'appartenenza a una lista può essere valutata utilizzando l'operatore `in`

Le liste sono **mutabili**, il che significa che il loro elemento può essere modificato una volta definito.



```
list1 = [1, 'two', 3.14, 0]

# membership can be tested using the in operator
1 in list1
>>> True
'five' in list1
>>> False
```



```
list1 = [1, 'two', 3.14, 0]
list2 = [2, 4, list1, True]

# element of a list can be modified
list1[2] = 2.72
print(list1)
>>> [1, 'two', 2.72, 0]

# also list2 has changed!
print(list2)
>>> [2, 4, list1, True]
```



List Methods

Python comes from numerous built-in list methods

Method	Description
<code>append(element)</code>	Append element at the end of the list
<code>extend(list2)</code>	Extend the list with the elements from list2
<code>index(element)</code>	Return the lowest index of the list containing element
<code>insert(index, element)</code>	Insert element at index index
<code>pop()</code>	Remove and return the last element from the list
<code>reverse()</code>	Reverse the list in place
<code>remove(element)</code>	Remove the first occurrence of element from the list
<code>sort()</code>	Sort the list in place
<code>copy()</code>	Return a copy of the list
<code>count(element)</code>	Return the number of elements equal to element in the list

[5. Data Structures — Python 3.12.2 documentation](#)



List Methods



```
# some list methods at play
```

```
q = []
```

```
q.append(4)
```

```
q.extend([6, 7, 8])
```

```
q.insert(1, 5)
```

```
q.remove(7)
```

```
q.index(8)
```

EXERCISE: try these methods on an empty list in your notebook and check the output



List Methods – Example



```
# using the methods append and pop to make a  
stack
```

```
stack = []  
stack.append(1)  
stack.append(2)  
stack.append(3)  
stack.append(4)  
print(stack)  
>>> [1, 2, 3, 4]  
stack.pop()  
print(stack)  
>>> [1, 2, ,3]
```

Exercise

Il metodo `split()` genera una lista di sottostringhe da una data stringa, suddivisa in base a un separatore specificato (spazio vuoto predefinito). Utilizza il metodo sulla stringa `'Jan Feb Mar Apr May Jun'` per

1. Contare il numero di mesi nella stringa (il metodo `len()` funziona anche sulle liste!)
2. Accesso al terzo mese della stringa
3. Invertire l'ordine dei mesi



Tuples

Una tupla è un elenco immutabile, il che significa che i suoi elementi non possono essere assegnati

```
● ● ●  
  
# tuples are constructed by using parentheses  
t = (1, 'two', 3.)  
print(t)  
>>> (1, 'two', 3.)  
  
t[1]  
>>> 'two'  
t[2] = 4  
>>> Traceback (most recent call last):  
>>> File "<stdin>", line 1, in <module>  
>>> TypeError: 'tuple' object does not support  
item assignment
```



Tuples

Le tuple sono utili per fare *packing* e *unpacking* insieme di elementi

```
# tuple packing
t = 1, 2, 3
print(t)
>>> (1, 2, 3)

# tuple unpacking
a, b, c = 97, 98, 99
print(b)
>>> 98
```

N.B. solo i metodi `count` e `index` sono disponibili per le tuple

ESERCIZIO: utilizzare l'impacchettamento e l'unpacking delle tuple per scambiare i valori di due variabili `a`, `b`



Loops

Per accedere agli elementi di un array (oggetto iterabile) uno alla volta, è necessario un ciclo for

```
fruit_list = ['apple', 'melon', 'banana', 'orange']  
  
# to define a for loop use the following syntax  
for fruit in fruit_list:  
    print(fruit)  
  
>>> apple  
>>> melon  
>>> banana  
>>> orange
```



Loops

È possibile eseguire un ciclo su qualsiasi oggetto iterabile: una stringa, un array, una tupla. Vedremo più avanti altri esempi di iterabili.

```
# we can also iterate over string
single_fruit = 'apple'
for letter in fruit:
    print(letter, end='.')

>>> a.p.p.l.e.

# loops can be nested
fruit_list = ['apple', 'melon', 'banana', 'orange']
for fruit in fruit_list:
    for letter in fruit:
        print(letter, end='.')
    print()

>>> a.p.p.l.e.
>>> m.e.l.o.n.
>>> b.a.n.a.n.a.
>>> o.r.a.n.g.e.
```



Loops methods

- range è un metodo utile per costruire sequenze numeriche iterabili per cicli

`range([a0=0], n, [stride=1])`

ESERCIZIO: Utilizzando il metodo dell'intervallo, calcola i primi 100 numeri di Fibonacci

```
# range allows to iterate over
# a sequence of progressing numbers
for x in range(5):
    print(x)

>>> 0
>>> 1
>>> 2
>>> 3
>>> 4

for range(1, 5):
    print(x)

for x in range(0, 6, 2):
    print(x)

for x in range(10, 0, -2):
    print(x)
```



Loops methods

Anche questa volta Python si presenta con utili metodi di loop integrati:

- `enumerate(list)`
restituisce una tupla con l'indice e l'elemento della lista
- `zip(lists)`
restituisce una tupla con ogni elemento di due o più liste

```
mammals = ['kangaroo', 'panda', 'monkey']
for i in range(len(mammals)):
    print(i, ':', mammals[i])

# this can be obtained more pythonically
# with enumerate
for i, mammal in enumerate(mammals):
    print(i, ':', mammal)

# you can iterate over two (or more) sequences
# at the same time
plants = ['bamboo', 'rose', 'aloe']

for plant, mammal in zip(plants, mammals):
    print(plant, 'is a plant', mammal, 'is a mammal')
```


Exercises

1. La distanza di Hamming tra due stringhe di uguale lunghezza è il numero di posizioni in cui i caratteri sono diversi. Scrivi una routine Python per calcolare la distanza di Hamming tra due stringhe, s1 e s2.
2. La funzione fattoriale $n! = n \cdot (n - 1) \cdot (n - 2) \dots 3 \cdot 2 \cdot 1$ è il prodotto dei primi n numeri interi positivi. Scrivi una routine per calcolare il fattoriale in Python.

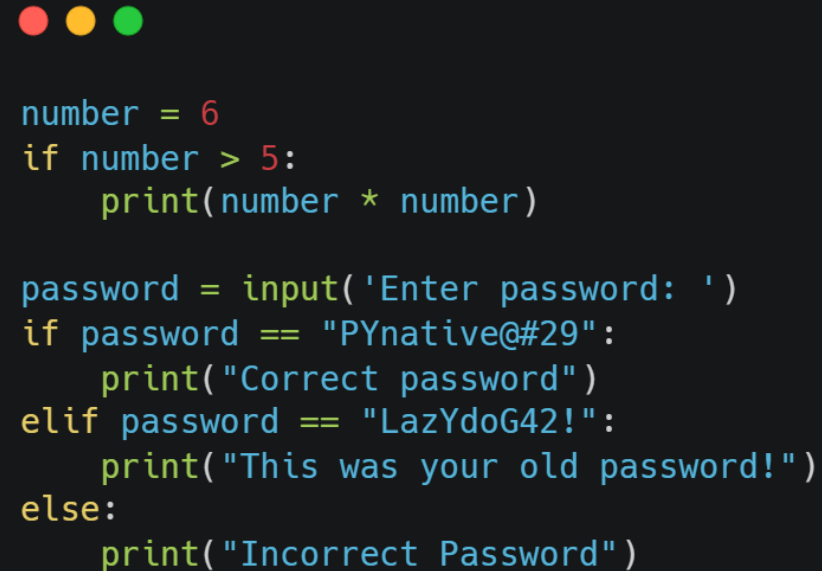


If...else statement

Di solito è necessario che un blocco di codice venga eseguito in modo condizionale sulla base di alcuni controlli. Questo si ottiene con l'istruzione if.

```
if condition:
    # Code block executed if condition is True
    statement1
    statement2
    # ...

if condition1:
    # Code block executed if condition1 is True
    statement1
elif condition2:
    # Code block executed if condition2 is True
    statement2
else:
    # Code block executed if condition1 and condition2 are
    False
    statement
```



```
number = 6
if number > 5:
    print(number * number)

password = input('Enter password: ')
if password == "PYnative@#29":
    print("Correct password")
elif password == "LazYdoG42!":
    print("This was your old password!")
else:
    print("Incorrect Password")
```



While loop

Il ciclo while esegue un blocco di codice finché la condizione viene soddisfatta

```
while condition:
    # Code block executed while condition is True
    statement1
    statement2
    # ...
```

A dark-themed terminal window with three colored window control buttons (red, yellow, green) at the top left. It contains Python code for a while loop that counts from 0 to 4 and prints the count.

```
count = 0
while count < 5:
    print("Count:", count)
    count += 1

print(count)
>>> 5
```



Example

```
print("Welcome to the Coffee Shop!")

order_loop = True

while order_loop:
    print("what do you want to order? Insert the order number")
    print("1. Regular Coffee")
    print("2. Cappuccino")
    print("3. Macchiato")

    order_number = input()

    if order_number == '1':
        print("You ordered regular coffee. Coming")
    elif order_number == '2':
        print("You ordered cappuccino. Coming")
    elif order_number == '3':
        print("You ordered macchiato. Coming")
    else:
        print("Please insert a valid order number")

    print("Do you want to place another order? y/n")

    another_order = input()
    if another_order == 'n':
        order_loop = False
```



Exercises

L'*algoritmo di Luhn* è una formula di checksum utilizzata per convalidare i numeri di carte di credito e conti bancari. È progettato per prevenire errori comuni nella trascrizione del numero e rileva tutti gli errori a singola cifra e quasi tutte le trasposizioni di due cifre adiacenti.

L'algoritmo può essere scritto come segue:

1. Inverti il numero: Considera il numero di carta di credito come una sequenza di cifre e inverti l'ordine di queste cifre.
2. Raddoppia le cifre a posizione pari (indici dispari): Partendo dalla prima cifra (quella originariamente all'ultima posizione) del numero invertito, raddoppia il valore di ogni cifra che si trova in una posizione pari (seconda cifra, quarta cifra, sesta cifra, ecc.).
3. Se una cifra raddoppiata risulta in un numero maggiore di 9, somma le due cifre (ad esempio, la cifra 6 diventa 12 e quindi $1+2 = 3$).
4. Somma tutte le cifre: Somma tutte le cifre del numero modificato (incluse quelle che non sono state raddoppiate e quelle che sono state modificate al passo 3).
5. Se la somma dell'array modulo 10 è 0, il numero di carta di credito è valido.

Scrivi un programma Python che accetti un numero di carta di credito come una stringa di cifre (possibilmente a gruppi, separate da spazi) e stabilisca se è valido o meno. Ad esempio, la stringa "4799 2739 8713 6272" è un numero di carta di credito valido, ma qualsiasi numero con una singola cifra in questa stringa non viene modificato.



Exercises

Obiettivo: Scrivere uno script in Python per guidare un giocatore (P) attraverso un labirinto testuale 2D, dalla partenza (S) fino all'uscita (E).

Le Regole:

- Il tuo mondo è una lista 2D (una lista di liste).
- # è un muro.
- (spazio) è un sentiero aperto.
- S è la partenza (Start).
- E è l'uscita (Exit).



Exercises

Hai già tutte le competenze necessarie. Questo progetto userà:

- Variabili: Per salvare la posizione del giocatore (riga e colonna).
- Liste (2D): Per creare la struttura del labirinto.
- Cicli (while): Per creare il motore principale del gioco che continua a girare.
- Cicli (for): Per analizzare la struttura del labirinto.
- Istruzioni Condizionali (if/elif/else): Per gestire il movimento e controllare le collisioni con i muri.

Exercises

Liste 2D (Liste di liste)

Immagina una lista 2D come una griglia o una tabella. È una lista che contiene altre liste come suoi elementi. Nel nostro gioco, ogni lista interna rappresenta una riga del labirinto.

```
maze = [  
    ["#", "#", "#"], # Riga 0  
    ["#", "S", " "], # Riga 1  
    ["#", "#", "E"]  # Riga 2  
]  
  
# Per accedere a 'S': maze[1][1] (riga 1, colonna 1)
```



Exercises

Cicli for Annidati

Per "visitare" ogni casella di una griglia 2D, abbiamo bisogno di due cicli for, uno dentro l'altro. Il ciclo esterno scorre le righe. Il ciclo interno scorre le colonne (gli elementi) di ogni riga.

```
# Esempio: trovare le coordinate della 'S'
for riga_index, riga in enumerate(maze):
    for colonna_index, elemento in enumerate(riga):
        if elemento == "S":
            print(f"Trovata la S alla riga {riga_index}, colonna {colonna_index}")
```



Exercises

Passaggi

- Crea il Mondo: Definisci il tuo maze usando una lista 2D.
- Trova il Giocatore: Usa un ciclo for annidato per trovare le coordinate di 'S' e salvarle in player_row e player_col.
- Costruisci il Game Loop: Crea un ciclo while True:.
- All'interno del Ciclo:
 - Pulisci lo schermo e stampa il labirinto con il giocatore (P).
 - Controlla se il giocatore ha raggiunto 'E'. Se sì, vinci e interrompi il ciclo.
 - Chiedi all'utente l'input (w/a/s/d).
 - Controlla se la mossa desiderata è valida (non è un muro).
 - Se è valida, aggiorna le coordinate del giocatore.



Exercises

Bonus

- Aggiungi le Trappole: Aggiungi un carattere 'T' nel labirinto. Se il giocatore ci finisce sopra, ha perso.
- Crea un Teletrasporto: Aggiungi una casella speciale 'O'. Quando il giocatore ci arriva, viene teletrasportato in un altro punto vuoto casuale.
- Fantasma Errante: Aggiungi un nemico 'G' che si muove in una direzione casuale dopo ogni mossa del giocatore. Se ti cattura, hai perso.

Suggerimento: Avrai bisogno di usare la libreria random per queste sfide!



Functions

Una funzione è un blocco di codice raggruppato e nominato per eseguire un'attività specifica.

- Consente di riutilizzare il codice senza doverlo replicare in parti diverse del programma.
- Consente di suddividere attività complesse in procedure separate, migliorando la leggibilità e la manutenzione del codice.

```
def square(x):  
    x_sqrd = x ** 2  
    return x_sqrd  
  
n = 2  
n_sqrd = square(n)  
  
print(n_sqrd)  
>>> 4  
  
print("The square of {} is {}".format(n, square(n)))  
>>> The square of 2 is 4
```

EXERCISE: Make the Lhun Alghoritm example code a function



Functions

- Gli argomenti della funzione sono indicati tra parentesi e possono essere più di uno.
- Per restituire due o più valori da una funzione, è possibile impacchettarli in una tupla.
- Il valore restituito non è obbligatorio. Se non specificato esplicitamente, la funzione restituisce None.



```
import math

# function to return the roots of a*x**2 + b*x + c
def roots(a, b, c):
    delta = b ** 2 - 4 * a * c
    r1 = (-b + math.sqrt(delta)) / 2 / a
    r2 = (-b - math.sqrt(delta)) / 2 / a
    return r1, r2

print(roots(1., -1., -6.))
>>> (3.0, -2.0)
```



```
# function to greet a person
def greet(name):
    print("Greetings to {}".format(name))

print(greet('Claudio'))
>>> Greetings to Claudio!

type(greet('Claudio'))
>>> <class 'NoneType'>
```



Positional and Keyword arguments

- Gli argomenti passati nell'ordine in cui vengono forniti sono chiamati argomenti posizionali.
- Gli argomenti possono essere passati anche in un ordine arbitrario impostandoli esplicitamente, e sono chiamati argomenti keyword.
- Gli argomenti keyword devono sempre seguire quelli posizionali.

```
# positional arguments
roots(1., -1., 6.)
# keywords arguments
roots(a=1., c=6., b=-1)
# you can mix them
roots(1., b=-1, c=6.)
# but keywords must always
# go after positional
roots(b=-1., 1., -6)
>>> File "<stdin >", line 1
SyntaxError: non-keyword arg after keyword arg
```



Scope

Una funzione può definire e utilizzare le proprie variabili.

- In questo caso, tali variabili sono locali e non sono disponibili all'esterno della funzione.
- Le variabili definite al di fuori delle definizioni di funzione sono globali e sono accessibili alle funzioni.
- La ricerca avviene prima nell'ambito locale: se la variabile locale ha lo stesso nome di una variabile globale, la funzione prende quella locale.

```
def func1():  
    a = 5  
    print(a, b)
```

```
b = 6  
func2()  
>>> 5 6
```

```
def func2():  
    a = 5  
    print(a)
```

```
a = 6  
func2()  
>>> 5  
print(a)  
>>> 6
```



Funzioni ricorsive

Una funzione che può chiamare se stessa è detta *ricorsiva*. Non è spesso necessaria, ma può portare a una notevole semplificazione in alcune situazioni.

Una situazione classica è il fattoriale:

$$n! = n \cdot (n - 1) \cdot (n - 2) \dots \cdot 2 \cdot 1$$

```
# compute the factorial of n
def factorial(n):
    if n == 1:
        return 1
    return n * factorial(n-1)

print(factorial(5))
```

✓ 0.0s

120



Argomenti

Gli argomenti delle parole chiave possono essere utilizzati per definire **argomenti facoltativi**. Se il chiamante non fornisce un valore per questo argomento allora è usato un valore predefinito.

```
def report_length(value, units='m'):  
    return 'The length is {:.2f} {}'.format(value, units)  
  
report_length(33.136, 'ft')  
>>> 'The length is 33.14 ft'  
report_length(10.1)  
>>> 'The length is 10.10 m'
```

Stringhe di documentazione

- Una *stringa di documentazione* è una stringa letterale che spiega cosa fa la funzione, descrivendo suo comportamento .
- Stringa di documentazione Dovrebbe fornire dettagli su come usare la funzione : quale gli argomenti passano alla funzione e quale oggetti ritorna.



```
def add_binary(a, b):  
    """  
    Returns the sum of two decimal numbers in binary  
    digits.  
  
    Parameters:  
        a (int): A decimal integer  
        b (int): Another decimal integer  
  
    Returns:  
        binary_sum (str): Binary string of  
    the sum of a and b  
    """  
    binary_sum = bin(a+b)[2:]  
    return binary_sum  
  
print(add_binary.__doc__)
```



Esempio

```
import math

# Define a function that takes a list of numbers and
# returns the mean, median, and mode
def stats(numbers, offset=0.0):
    """
    This function returns the mean, median, and mode of a
    list of numbers.

    Parameters
    -----
    numbers : list of int or float
        The list of numbers to compute the statistics for.

    Returns
    -----
    mean : float
        The mean of the numbers.
    std_dev : float
        Standard deviation of the numbers.
    """
    tot_sum = 0
    for number in numbers:
        tot_sum += number

    mean = tot_sum / len(numbers)
    mean += offset

    sigma = 0
    for number in numbers:
        sigma += (number-mean) ** 2

    std_dev = math.sqrt(sigma / len(numbers))

    return mean, std_dev
```



PEP 484: Type Hints

I Type Hints (Python 3.5) consentono agli utenti di annotare le propri funzioni per indicare i tipi di input e output. Questo È utile per l'analisi statica e per la scrittura di codice privo di bug.

```
def count_vowels(x: str, include_y: bool = False) -> int:
    """Returns the number of vowels contained in x"""
    vowels = set("aeiouAEIOU")
    if include_y:
        vowels.update("yY")

    count = 0
    for char in x:
        if char in vowels:
            count += 1

    return count
```



Esercizi

1. Scrivi due funzioni che , dato due liste di lunghezza 3 rappresentanti tre dimensionale vettore **a** e **b** , calcola il prodotto scalare **$a \cdot b$** e il prodotto vettoriale **$a \times b$** .
2. Scrivi una funzione per calcolare il saldo di un conto di risparmio con un tasso di interesse annuale fisso per un certo numero N di anni (argomento facoltativo , predefinito 1). Calcola il saldo dopo 4 anni al tasso del 5%, a partire da 100€.
3. Scrivi una funzione chiamata `describe_city()` che prende in input il nome di una città e la sua nazione. Fa una verifica sul tipo dell'input e restituisce in output una stringa formattata bene, con la prima lettera della città e della nazione in maiuscolo, più una descrizione (se passata alla funzione).



Esercizi

Obiettivo: RIFATTORIZZARE il codice di `labyrinth.py` utilizzando le funzioni

Le funzioni rendono il codice migliore sotto molti aspetti:

1. DRY (Don't Repeat Yourself - Non Ripeterti): Se devi fare la stessa cosa più volte (es. stampare il labirinto), scrivi il codice una sola volta dentro una funzione e richiamala quando serve.
2. Leggibilità: Dare un nome a un blocco di codice (es. `display_maze`) rende subito chiaro cosa fa quella parte del programma.
3. Modularità: Suddividono un problema complesso (il gioco) in sotto-problemi più piccoli e semplici da gestire (mostrare la mappa, muovere il giocatore, etc.).
4. Manutenzione Semplice: Se devi correggere un errore nella logica di movimento, modifichi solo la funzione `move_player`, senza toccare il resto.

Esercizi

Possiamo identificare delle parti con un compito specifico?

```
# ... setup del labirinto ...

# BLOCCO 1: TROVARE LA 'S'
player_row, player_col = 0, 0
for r_index, riga in enumerate(maze):
    # ... logica per trovare 'S' ...

# BLOCCO 2: GAME LOOP
while True:
    # BLOCCO 3: MOSTRARE IL LABIRINTO
    os.system('clear')
    # ... logica per stampare il labirinto con 'P' ...

    # ... controlla la vittoria ...

    # BLOCCO 4: GESTIRE IL MOVIMENTO
    move = input(...)
    # ... logica per calcolare la nuova posizione ...
    # ... controllo delle collisioni ...
```



Esercizi

Immagina di lavorare per un'azienda cloud e di dover scrivere una funzione Python per configurare e "avviare" un nuovo server virtuale per un cliente. I clienti possono scegliere diverse configurazioni, alcune obbligatorie e altre facoltative. Il tuo compito è creare una funzione `configura_server()` che accetti vari parametri per personalizzare il server.

Requisiti della Funzione

La funzione `configura_server()` deve accettare:

- Un nome per il server (argomento posizionale obbligatorio).
- Un sistema operativo (un altro argomento posizionale obbligatorio).
- La RAM in GB, che di default è 8 (argomento con valore di default).
- [Una lista di pacchetti software da installare (un numero arbitrario di argomenti posizionali).]
- [Una serie di impostazioni di rete avanzate, come `ip_statico`, `firewall_rules`, `dominio`, ecc. (un numero arbitrario di argomenti keyword).]

Cosa Deve Fare la Funzione

La funzione deve stampare un riepilogo della configurazione del server in modo chiaro e leggibile.



Dizionari

Un dizionario è un array associativo (hash- map) che memorizza *i valori* a cui si può accedere attraverso *chiavi*

```
# a dictionary can be defined by giving
# key: value between braces

height = {
    'Burj Khalifa': 828,
    'One World Trade Center': 541.3,
    'Gran Torre Santiago': 300.
}

# an individual item can be retrieved
# by indexing it with its keys
# either as a string or by a variable

print(height['Gran Torre Santiago'])
>>> 300.

building = 'One World Trade Center'
print(height[building])
>>> 541.3
```

Qualunque oggetto immutabile può essere una chiave (stringa , numeri , tuple)



Modi per inizializzare un dizionario

- Tramite parentesi graffe
{chiave: valori }
- Indicizzazione
- Passando sequenza (chiave- valori
)
al dict

Appartenenza a un dizionario è
verificato con la parola chiave `in`

Il numero di chiavi è controllato con
`len`

```
# curly braces initialization
tel = {'jack': 4098, 'sape': 4139}

# directly assigning keys
tel['guido'] = 4127

# list on a dict return a list of the keys
print(list(tel))
>>> ['jack', 'sape', 'guido']

print('guido' in tel)
>>> True

# dict construct can build dictionaries directly
# from key-value sequences
tel = dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
```



Esempio

- Il dizionario è un oggetto iterabile :
il ciclo su un dizionario fornisce le chiavi (in ordine di inserimento)
- Le chiavi devono essere univoche ,
ma i valori possono non esserlo



```
# for loops over dictionaries return the keys
```

```
numerals = {'one': 'I', 'two': 'II', 'three': 'III'}
```

```
for i in numerals:  
    print(i, numerals[i])
```

```
# although the dictionary keys must be unique,  
# dictionary valuse need not be
```

```
numerals[1] = 'I'  
numerals[2] = 'II'  
numerals[3] = 'III'
```

```
print(numerals)
```

```
>>> {'one': 'I', 'two': 'II', 'three': 'III', 1: 'I', 2: 'II',  
3: 'III'}
```

```
for i in range(1, 4):  
    print(numerals[i], end=' ')4098))
```



Metodi

- `get (chiave)`
recupera il valore dal dizionario associato alla chiave.
- Se non presente, restituisce un valore predefinito (`None`)

```
mass = dict(Mercury=3.301e23, Venus=4.867e24, Earth=5.972e24)
mass['Earth']

# indexing a dictionary with a key that does not exist is an error
mass['Pluto']
```

```
-----
KeyError                                Traceback (most recent call last)
Cell In[3], line 5
      2 mass['Earth']
      4 # indexing a dictionary with a key that does not exist is an error
----> 5 mass['Pluto']
      7 print(mass.get('Pluto'))

KeyError: 'Pluto'
```

```
print(mass.get('Pluto'))
print(mass.get('Pluto', -1))
```

```
None
-1
```



Metodi del dizionario - 1

- `keys`, `values`, `items`

Questi metodi ritornano rispettivamente le chiavi del dizionario , la coppia key-value (COME tuple)

```
planets = mass.keys()
print(planets)
```

```
for planet in planets:
    print(planet, mass[planet])
```

```
# keys does not return a list
# if you want a list of the dict keys, you can use list
```

```
planet_list = list(mass.keys())
print(planet_list)
```

```
dict_keys(['Mercury', 'Venus', 'Earth'])
Mercury 3.301e+23
Venus 4.867e+24
Earth 5.972e+24
['Mercury', 'Venus', 'Earth']
```



Metodi del dizionario - 2

- keys, values, items

Questi metodi ritornano
rispettivamente le chiavi del
dizionario, la coppia key-value
(COME tuple)

```
# values retrieve the dict values
print(mass.values())

# items retrieve the dict pair (key, value)
print(mass.items())

# we can loop on the returned objects
for planet, mass in mass.items():
    print(planet, mass)
```

```
dict_items([('Mercury', 3.301e+23), ('Venus', 4.867e+24), ('Earth', 5.972e+24)])
dict_values([3.301e+23, 4.867e+24, 5.972e+24])
Mercury 3.301e+23
Venus 4.867e+24
Earth 5.972e+24
```



Keywords arguments

Python include funzionalità per gestire il caso in cui una funzione non sa quali argomenti può ricevere :

- `*args` dopo posizionale l'argomento pone qualsiasi argomento posizionale aggiuntivo nella tupla `args`
- `**kwargs` impacchetta argomenti aggiuntivi per parole chiave non specificati nella definizione di funzione in un dizionario `kwargs`

```
def func(a, b, *args):  
    print(args)  
  
func(1, 2, 3, 4, 'prova')
```

✓ 0.0s

(3, 4, 'prova')

```
def func(a, b, **kwargs):  
    for k in kwargs:  
        print(k, '=', kwargs[k])  
  
func(1, 2, c=3, d=4, s='prova')
```

✓ 0.0s

c = 3
d = 4
s = prova



Set

Un insieme è un *insieme non ordinato* di *pezzi unici*. Esso è utile per rimuovere duplicati da una sequenza

```
s = set([1, 1, 3, 2, 2, 4])
print(s)
>>> {1, 3, 2, 4}

# cardinality of the set
len(set)
>>> 4

# membership
2 in s
>>> True

# loop
for item in s:
    print(item)
>>> 1
>>> 3
>>> 2
>>> 4
```

<https://docs.python.org/3/library/stdtypes.html#set>



Set

I set sono anche utili per le comuni operazioni matematiche sugli insiemi

```
A = set((1, 2, 3))
B = set((1, 2, 3, 4))

print(A.issubset(B))

print(A.union(B))
print(A | B)

print(A.intersection(B))
print(A & B)
```

True
{1, 2, 3, 4}
{1, 2, 3, 4}
{1, 2, 3}
{1, 2, 3}

ESERCIZIO: Scrivi una funzione, utilizzando oggetti set, per rimuovere i duplicati da un elenco ordinato
ad esempio [1, 1, 2, 3, 4, 4, 5, 7, 8, 8, 9]

Domande e risposte

Prevedi e spiega l' effetto delle seguenti affermazioni :

```
set('hellohellohello')  
set(['hellohellohello'])  
set(('hello', 'hello', 'hello'))
```

Esercizi

1. Scrivi una funzione , usando gli insiemi, per rimuovere duplicati da un elenco ordinato .
2. Utilizzare il dizionario dei simboli del codice Morse disponibile nella cartella, per scrivere un programma che può tradurre un messaggio in codice Morse, utilizzando spazi per delimitare singole lettere del codice Morse e barre / per delimitare le parole.

CONFIDENTIALITY NOTICE

Le informazioni, i dati e le immagini contenuti in questo documento sono strettamente confidenziali e riservati, di esclusiva proprietà di Lutech.

Sono disponibili esclusivamente per persone o società a cui è stato direttamente consegnato il documento e non sono divulgabili a terzi senza il consenso scritto dell'Ufficio Comunicazione di Lutech, che può essere contattato all'indirizzo comunicazione@lutech.it.

I loghi di terze parti (es. partner, clienti, ecc.) sono da considerarsi indicativi.



**Headquarter**

Via Massimo Gorki, 30/32C
20092 Cinisello Balsamo (MI)
P.IVA 02824320176
Tel +39 02 25427011
Fax +39 02 25427090

Sede Roma

Via Via Giuseppe Grezar, 34
00142 Roma (RM)

Sede Legale

Via Dante, 14
20121 Milano (MI)

www.lutech.group