

Espressioni e numeri

- Espressioni
- Valutazione
- Numeri (int e float)
 - i numeri sono **oggetti**
 - Tipo (o classe) di un numero (funzione type)
- La divisione restituisce sempre un numero con la virgola, le altre dipende, se sono tutti interi restituisce un intero, altrimenti un numero con la virgola
- Operatori aritmetici (+, -, *, /)
- Regole di precedenza
- Parentesi tonde
- Altri operatori (**, //, %)
- Regole di precedenza di questi nuovi operatori (** precedenza massima, divisione intera e resto come il *)

Esercizi

Area del trapezio

Sapendo che l'area del trapezio si calcola in questo modo:

$$\frac{(base_{maggiore} + base_{minore}) \cdot altezza}{2}$$

calcola l'area del trapezio avente le seguenti misure:

$$base_{maggiore} = 14.8$$

$$base_{minore} = 10.9$$

$$altezza = 6.1$$

Quanti mesi hai?

Calcola quanti mesi di vita hai vissuto

Tradurre le seguenti espressioni aritmetiche

1. Moltiplica per 4 la differenza fra il quadrato di 5 e il quadrato di 3
2. Eleva al quadrato la somma del cubo di 2 e del quadrato di 1, poi dividi il risultato per 27

Espressioni che non resituiscono numeri

- Le espressioni aritmetiche resituiscono oggetti di tipo numerico (int e float)
- Ma ci sono anche espressioni che restituiscono oggetti di tipo diverso
- Confrontiamo due valori verificando che uno sia maggiore dell'altro
- Il tipo **bool**
- Operatori di confronto (>, <, ==, <=, >=, !=)
- Si possono confrontare valori, ma anche espressioni:
 - $(4 * 5 + 3) == (7 + 16)$

Esercizi

- Scrivere un'espressione che restituisca **True**
- Scrivere un'espressione che restituisca **False**

Variabili

- Necessità di salvare i dati da qualche parte
- variabili ben formate:
 - nome (lettere, numeri, _)
 - simboli ammessi
 - parole riservate
 - lunghezza
 - convenzioni
- operatore di assegnamento (=)
 - tramite questo operatore associamo un oggetto (un numero o *qualunque* altro oggetto) a una variabile
 - assegnazioni multiple
- cosa possiamo assegnare ad una variabile?
- l'assegnamento avviene da destra a sinistra, prima python *valuta* l'espressione a destra, poi assegna il valore valutato alla variabile
- in python non esistono variabili propriamente dette, ma **etichette**
- il valore associato a una variabile può cambiare, il suo nome no!
- operazioni fra variabili
- stampa di un valore tramite **print()**

Esercizi

assegnamento e operazioni fra variabili

Assegna a 2 variabili chiamate **base** e **altezza** i seguenti valori: 12.8 e 13.6 poi calcola l'area del rettangolo e assegnala alla variabile **area**. Stampa il risultato con la funzione **print()**

Area del Trapezio

Calcola l'area del trapezio dell'esercizio precedente assegnando i valori alle variabili **base_maggiore**, **base_minore**, **altezza** e infine il calcolo dell'area alla variabile **area**

Usa le variabili per risolvere il seguente problema

Se un violino costa 500 euro, quanto lo pagherò se ottengo uno sconto del 30%?

Le Stringhe

- Cosa sono le stringhe?
- il tipo **str**
- Tipi di delimitatori (",')
- Includere un delimitatore in una stringa
- operatori sulle stringhe (*, +)
- usare print() con le stringhe
- caratteri speciali (\n, \t)

Esercizi

Scrivi le istruzioni per stampare 5 volte la stringa "Buon Compleanno"

Scrivi la sequenza di istruzioni per ottenere il perimetro e l'area di un rettangolo che abbia i lati di cm 9 e cm 5 e infine stampa la seguente stringa: "Dato un rettangolo con lati 10 e 5, il perimetro sarà uguale a ... e l'area a ...". Naturalmente sostituisci i puntini con i giusti valori

Introspezione

Python consente di fare introspezione, cioè di ottenere informazioni sugli oggetti presenti in un programma.

Importanti funzioni di introspezione:

- identità di un oggetto: **id()**
- tipo di un oggetto: **type()**
- elenco degli attributi: **dir()**
- descrizione di una funzione o di un attributo: **help()**

Ogni tipo definisce un insieme di attributi, alcuni dei quali sono dati *accessorii* dell'oggetto, altri sono *chiamabili*, cioè sono delle vere e proprie operazioni da compiere sull'oggetto. Questi attributi *operazionali* sono detti **metodi**. Per sapere se un attributo di un oggetto è un metodo, si può chiamare l'attributo con la funzione *callable*:

```
callable(str.capitalize)
```

che resituirà **True** o **False**.

Metodi importanti delle stringhe

Alcuni metodi restituiscono informazioni su una stringa:

- `upper()`: restituisce una stringa tutta maiuscola
- `lower()`: restituisce una stringa tutta minuscola
- `replace()`: sostituisce uno o più caratteri con altri
- `count()`: conta il numero di occorrenze di un caratteri o di una sequenza di caratteri
- `isdigit()`: restituisce **True** se la stringa contiene delle cifre

```
# str.upper()
stringa = "sheldon cooper"
print(stringa.upper())
>>> SHELDON COOPER
```

```
# str.lower()
stringa = "ShEldOn cooPer"
print(stringa.lower())
>>> sheldon cooper
```

```
# str.replace()
stringa = "sheldon cooper"
```

```
print(stringa.replace('o','a'))
>>> sheldan caaper
```

```
# str.count()
stringa = "sheldon cooper"
print(stringa.count('o'))
>>> 3
```

```
# str.isdigit()
stringa = "sheldon cooper"
print(stringa.isdigit())
>>> False
```

```
# str.isdigit()
stringa = "23"
print(stringa.isdigit())
>>> True
```

I metodi **NON** modificano la stringa originale, ma creano una **NUOVA** stringa modificata. Gli oggetti di tipo **str** infatti sono **IMMUTABILI**, come del resto gli **int** e i **float**.

Funzioni importanti delle stringhe

input()

Permette di *catturare* una sequenza di caratteri immessi da tastiera. Permette di stampare una stringa di descrizione del testo da immettere e restituisce a sua volta la stringa immessa da tastiera.

```
input("scrivi il tuo nome: ")
```

len()

Restituisce la lunghezza di una stringa, cioè la quantità di caratteri che la compongono.

```
stringa = "Leonard Hofstadter"
lunghezza = len(stringa)
print(lunghezza)
>>> 17
```

Conversione

La conversione di *tipo* permette di convertire un oggetto di un tipo in un oggetto di un altro tipo. Ad esempio un numero di tipo *float* può essere convertito in un oggetto di tipo *int*:

```
>>> numero = 13.8
>>> numero_convertito = int(numero)
>>> print(numero_convertito)
>>> 13
```

Le stringhe possono essere convertite in numeri. Se una stringa contiene delle cifre, possiamo convertirla in numero intero con **int()**. Se invece contiene cifre separate da un punto, possiamo convertirla in **float**.

```
stringa = '13'
print(type(stringa))
>>> <class 'str'>
numero = int(stringa)
print(numero)
print(type(numero))
>>> 13
>>> <class 'int'>
```

Il meccanismo della conversione consente di ottenere valori numerici da tastiera tramite la funzione **input()** e di convertirli con **int()** e **float()** in un valore di tipo numerico.

```
>>> anno_di_nascita = input("quando sei nato? ")
>>> anno_attuale = 2021
>>> eta = anno_attuale - anno_di_nascita
>>> print("Hai", eta, "anni!")
```

Esercitazioni

1. Acquisisci una stringa con la funzione **input()** e stampa la stringa acquisita, tutta al maiuscolo
2. Acquisisci due stringhe (una alla volta) con la funzione **input()** e stampa le due stringhe concatenate con l'operatore **+**
3. Scrivi un programma che prenda in input la base e l'altezza di un triangolo e restituisca l'area. Usa **input()** per acquisire i due valori, inseriscili in due variabili e convertili in numero, poi calcola l'area, inseriscila in una variabile **area** e stampa il suo valore
4. Scrivi un programma che acquisisca una stringa da input, stampi il tipo dell'oggetto, l'identità e l'elenco dei metodi
5. Esplora i metodi per le stringhe con la funzione **dir()** e cerca di capire cosa fanno i metodi **str.capitalize()**, **str.title()**, **str.strip()**. Usa la funzione **help()** su un metodo per accedere alla sua documentazione

Slicing

Una stringa è una sequenza *ordinata* di caratteri. In quanto ordinata i caratteri hanno una *posizione* identificata da un **indice**

stringa	'h'	'o'	'w'	'a'	'r'	'd'
indice	0	1	2	3	4	5

Slicing semplice (index)

Per accedere a un singolo carattere di una stringa si usa la notazione a **slicing** semplice:

```
str[index]
```

```
stringa = 'howard'
print(stringa[0])
>>> h
print(stringa[1])
>>> o
# posso usare l'indice -1 per ottenere l'ultimo carattere
print(stringa[-1])
>>> d
```

Slicing con inizio e fine

Con lo stesso tipo di notazione posso ottenere *fettine* di una stringa, cioè frammenti, definendo un punto di inizio e un punto di fine separati da due punti:

```
str[start:end]
```

```
stringa = 'howard'
# Ottengo tutti i caratteri dall'indice 1 all'indice 3 (escluso)
print(stringa[1:3])
>>> ow
```

Se si omette il punto di inizio, è implicito che il punto di inizio sia il primo carattere:

```
stringa = 'rajesh'
# Ottengo i caratteri dall'inizio fino all'indice 4 (escluso)
```



```
print(stringa[:4])
>>> raje
```

Se si omette il punto di fine, è implicito che il punto di fine sia l'ultimo carattere:

```
stringa = 'rajesh'
# Ottengo i caratteri dall'indice 4 fino alla fine
print(stringa[4:])
>>> sh
```

Slicing con inizio, fine e step

Si può aggiungere anche un terzo argomento, che definisce lo *step*, cioè il passo con cui estrarre i caratteri, la notazione è:

```
str[start:end:step]
```

```
stringa = "sheldon cooper"
# Ottengo i caratteri dall'indice 1 all'indice -1 (escluso) procedendo a
passi di due
print(stringa[1:-1:2])
>>> hlo op
```

Naturalmente è possibile omettere l'inizio e/o la fine:

```
stringa = "sheldon cooper"
# Ottengo i caratteri dall'inizio fino all'indice 6 (escluso) procedendo a
passi di due
print(stringa[:6:2])
>>> sed
```

Trucchetto: se si omette l'inizio e la fine si ottiene una copia dell'intera stringa:

```
stringa = 'leonard hofstader'
print(stringa[:])
>>> leonard hofstader
```

E se si aggiunge l'argomento *step* a -1 si ottiene l'intera stringa al contrario:

```
stringa = 'leonard hofstader'  
print(stringa[::-1])  
>>> redatsfoh dranoel
```

Esercitazioni

1. Scrivi un programma che prenda in input una stringa e stampi prima una stringa formata dai soli caratteri in posizione pari, poi un'altra stringa formata dai caratteri in posizione dispari
 - esempio:
 - stringa di entrata: "BigBangTheory"
 - uscita1: "Bgaghoy"
 - uscita2: "iBnTer"
2. Scrivi un programma che prenda in input una stringa e calcoli il numero dei soli caratteri pari (usa lo slicing e la funzione len())
3. Scrivi un programma che prenda in input una frase (una stringa formata da caratteri e spazi) e produca in uscita la frase senza gli spazi e stampi anche la lunghezza della stringa.
4. Scrivi un programma che prenda in input due stringhe (una alla volta), le concateni tramite l'operatore + e stampi 3 volte la nuova stringa ottenuta.
 - esempio:
 - stringa1: "ciao"
 - stringa2: "mondo"
 - uscita: "ciao mondo ciao mondo ciao mondo"
 - N.B. *Attenzione agli spazi!*

Selezione

Espressioni booleane

Si tratta di un'espressione booleana, cioè di un'espressione che, valutata, restituisce un valore pari a **True** o **False**. Un'espressione booleana può essere:

1. costituita da un solo elemento. In quel caso, alla valutazione, restituisce True se quell'elemento è un numero (int o float) diverso da 0 o un oggetto (stringa, lista, dizionario) non vuoti.
2. costituita da due operandi separati di un **operatore di relazione**:
 - <, minore di
 - >, maggiore di
 - ==, uguale a
 - !=, diverso da
 - <=, minore o uguale a
 - >=, maggiore o uguale a
3. costituita da due (o più) espressioni booleane *connesse* tramite **operatori logici**:
 - **and**
 - **or**
 - **not**

tavole di verità

A	B	and
V	V	V
V	F	F
F	V	F
F	F	F

A	B	or
V	V	V
V	F	V
F	V	V
F	F	F

A	not
V	F
F	V

Esempi

```
x = 10  
y = 15
```

valutare le seguenti espressioni:

```
x < 20  
x > 11  
x != y  
x == y  
x <= 10  
y >= 5
```

```
x > 9 and x < 11  
x > 5 or x < 9  
not x > 5
```

if

Istruzione che permette di eseguire un blocco di codice solo al verificarsi di una condizione

struttura

```
if condizione:  
    # esegui questo codice
```

tradotto in linguaggio naturale:

```
se la condizione è vera, allora fai una certa cosa
```

indentazione

Il codice python non *racchiude* i blocchi di codice fra parentesi graffe, come invece fanno il C, il java e il javascript. Per identificare un blocco di codice si usa l'*indentazione*. In una istruzione **if**, il blocco di codice da eseguire dopo i *due punti* deve essere scritto integralmente con lo stesso livello di indentazione:

```
if condizione:  
    print("A")  
    print("B")  
    print("C")
```

esempio

- valore assoluto di un numero inserito da tastiera

```
numero = int("inserisci un numero: ")  
  
if numero < 0:  
    numero = -numero  
  
print(numero)
```

esercizi

- acquisire 2 numeri interi e stampare la distanza in valore assoluto fra i due. Esempio:

```
a = 12  
b = 17  
>>> La distanza è 5
```


if/else

```
if condizione:
    # fai qualcosa
else:
    # fai un'altra cosa
```

esercizi

- acquisire due numeri (tramite input()) e stampare il maggiore fra i due
- acquisire un numero e stampare "è pari", se il numero è pari, altrimenti stampare "è dispari"
- chiedere all'utente di inserire una stringa e poi di inserire un numero n . Stampare quindi il carattere della stringa in posizione n . Se il numero n è più lungo della lunghezza della stringa, stampate invece la stringa "numero troppo alto"
- chiedere all'utente di inserire una stringa e stampare solo i caratteri che si trovano agli indici in posizione pari se il numero dei caratteri è pari, altrimenti stampare i caratteri agli indici dispari se il numero dei caratteri è dispari
- Chiedere all'utente di inserire un numero. Se il numero è minore o uguale a 100 stampare tanti asterischi (*) quanto vale il numero, altrimenti stampare tanti più (+) quanto vale il numero. Esempio, se il numero è 9, stampare:

```
*****
```

if/elif/else

struttura:

```
if condizione:
    # esegui questo blocco
elif condizione2:
    # esegui quest'altro blocco
else:
    # esegui quest'altro blocco ancora
```

Esempio

- trovare il maggiore fra 3 numeri interi inseriti dall'utente

```
primo = int(input("inserisci un numero: "))
secondo = int(input("inserisci un numero: "))
terzo = int(input("inserisci un numero: "))

if numero1 > numero2 and numero1 > numero3:
```

```
    print("il maggiore è", numero1)
elif numero2 > numero3:
    print("il maggiore è", numero2)
else:
    print("Il maggiore è", numero3)
```

annidare

```
numero1 = int(input("inserisci un numero: "))
numero2 = int(input("inserisci un numero: "))
numero3 = int(input("inserisci un numero: "))

if numero1 == numero2 == numero3:
    print("i numeri sono uguali")
else:
    if numero1 > numero2 and numero1 > numero3:
        print("il maggiore è", numero1)
    elif numero2 > numero3:
        print("il maggiore è", numero2)
    else:
        print("Il maggiore è", numero3)
```

if indipendenti

Nel caso del costrutto if/else e del costrutto if/elif/else le condizioni sono esclusive. Una condizione può essere vera o falsa, se è vera il resto del costrutto (elif o else) non viene preso in considerazione. Se la condizione è falsa il codice va alla successiva istruzione, se si tratta di un elif verifica la nuova condizione, se è vera ignora il resto del costrutto altrimenti va avanti.

A volte è necessario verificare condizioni a prescindere dal valore di verità di altre condizioni. Per questo si possono usare successivi costrutti if.

```
if condizione:
    # fai questo

if condizione2:
    # fai questo

if condizione3:
    # fai questo
```

In questo esempio se *condizione* è vera viene eseguito il blocco di codice sottostante, poi viene verificata la condizione successiva (*condizione2*) e così via.

esercizi

- Chiedere all'utente di inserire un numero. Se il numero è divisibile per 3 stampare "Il numero è divisibile per 3", se è divisibile per 2, stampare "Il numero è divisibile per 2", altrimenti stampare "Il numero non è divisibile né per 2 né per 3"

```
numero = 30
divisibile = 0

if numero % 2 == 0:
    divisibile += 2

if numero % 3 == 0:
    divisibile += 3

if divisibile == 0:
    print("il numero non è divisibile né per 2 né per 3")
elif divisibile == 2:
    print("il numero è divisibile per 2")
elif divisibile == 3:
    print("il numero è divisibile per 3")
else:
    print("il numero è divisibile per 2 e per 3")
```

ESERCITAZIONI

1. Lo scambio

Acquisire da input due valori, associarli alle variabili x e y e:

1. stamparne il contenuto
2. invertire i due valori
3. stamparne nuovamente il contenuto

esempio di output:

```
La variabile x vale 10
La variabile y vale 5
Effettuo lo scambio...
La variabile x vale 5
La variabile y vale 10
```

Python permette il riassegnamento contemporaneo di più variabili. Quindi è un'espressione valida la seguente:

```
x, y = y, x
```

Non usate questa tecnica per questo esercizio!!

2. Incremento (decremento)

A volte è necessario incrementare (o decrementare) il valore di una variabile, cioè prendere il valore della variabile, effettuare un'operazione e riassegnare il risultato alla variabile stessa.

```
x = 10
# incremento di 2 la variabile
x = x + 2
# ora x vale 12
# moltiplico per 3
x = x * 3
# ora x vale 36
# divido per 2
x = x / 2
# ora x vale 18.0
```

Questo meccanismo è talmente comune che esistono delle scorciatoie sintattiche:

- incremento con addizione: +=
- incremento con moltiplicazione: *=
- decremento con sottrazione: -=
- decremento con divisione: /=
- incremento con elevamento a potenza: **=

```
x = 10
# incremento di 2 la variabile
x += 2 # è uguale a scrivere x = x + 2
# moltiplico per 3
x *= 3
# divido per 2
x /= 2
print(x)
>>> 18.0
```

Esercizio

Scrivi un programma che prenda in input un numero intero e lo salvi in una variabile, poi incrementi il suo valore di 10, poi lo divida per 2 e stampi infine la variabile aggiornata.

Esempio:

```
Scrivi un intero: 14
La variabile number ora vale 14
Incremento di 10 e divido per 2...
La variabile number ora vale: 12.0
```

3. Somme e quadrati

Scrivi un programma che chieda tre numeri, ne calcoli la somma, la somma dei quadrati e il quadrato della somma. Infine, visualizza i risultati.

4. Controllo somma

Scrivi un programma che acquisisca 3 numeri interi, il terzo è la somma dei primi due. Se la somma è corretta stampare la stringa: "Bravo, sai fare le addizioni", altrimenti stampare la stringa: "Hai ancora da imparare! La somma è ...".

Esempio:

```
Inserisci il primo addendo: 16
Inserisci il secondo addendo: 13
```

```
Inserisci la somma: 25
>>> Hai ancora da imparare! La somma è 29
```

5. Estrazione caratteri

Scrivi un programma che prenda in input una stringa (parola o frase), elimini gli spazi e restituisca solo le **consonanti** se il numero totale dei caratteri è pari, altrimenti restituisca la stringa intera (privata degli spazi) al contrario e in maiuscolo.

Esempio:

```
Scrivi una frase: Ciao a tutti
>>> cttt

Scrivi una frase: Sheldon Cooper
>>> REPOOCNODLEHS
```

6. Accordi maggiori e minori (MIDI)

Il protocollo MIDI (di cui non ci occuperemo ora) rappresenta le altezze con un valore numerico compreso fra 0 e 127 (incluso) a partire dal Do-0. Così la nota 12 è il Do-1, il 24 è il Do-2. La nota 2 è il Re-0, la 14 è il Re-1 e così via.

Utilizzando il protocollo MIDI, acquisire una nota da input, controllare che rientri nel range consentito dal MIDI e, utilizzando quella nota come fondamentale, stampare in output:

- una triade maggiore
- una triade minore

esempio:

```
Inserisci la fondamentale: 60
triade maggiore: 60 64 67
triade minore: 60 63 67
```

7. Pitch class e MIDI

La *pitch class* (classe di altezze) è il nome di una nota privato della sua ottava di appartenenza. La nota **Do-6** è la nota Do della sesta ottava. La sua *pitch class* è **Do**. Ragionando in termini di protocollo MIDI, i multipli interi del 12, ossia tutte le note che, divise per 12, danno resto 0, appartengono alla *pitch class* **Do**, le note che, divise per 12, danno resto uguale a 1 appartengono alla *pitch class* **Do#**, e così via...

Scrivi un programma che prenda in input una nota (in formato MIDI) e restituisca la sua *pitch class* espressa come valore compreso fra 0 e 11 (incluso).

8. Intervalli

Scrivi un programma che:

1. Prenda in input 2 note MIDI
2. Estragga le relative *pitch class*
3. Classifichi (e stampi) l'intervallo fra le due *pitch class*
4. **Bonus track:** Stampi l'inverso dell'intervallo

Annotazioni: La classificazione degli intervalli nel sistema tonale dipende da "quale nome" diamo alle altezze: Do-Fa# è diverso da Do-Solb. In questo programma, per semplificare, stabiliamo una tavola degli intervalli a priori:

numero semitoni	intervallo
1	seconda minore
2	seconda maggiore
3	terza minore
4	terza maggiore
5	quarta giusta
6	quarta eccedente
7	quinta giusta
8	sesta minore
9	sesta maggiore
10	settima minore
11	settima maggiore