Corso di preparazione Python

Lezione 2

Aurora Arrus - 20/01/2024

Introduzione

Operatori e operandi

Gli ***operatori*** sono simboli che rappresentano calcoli.
Il valore a cui l'operatore è applicato si chiama ***operando***.

- **+**, addizione
- **-**, sottrazione
- ***, moltiplicazione
- **/**, divisione in virgola mobile
- **//**, divisione intera
- ****, potenza

```
>>> x = 7
>>> x / 2
3.5
>>> x // 2
3
```

Operatore modulo

L'operatore ***modulo*** è rappresentato dal simbolo **%** e restituisce il *resto della divisione tra l'operando di sinistra e quello di destra*.

```
>>> quoziente = 7 // 3
>>> quoziente
2
>>> resto = 7 % 3
>>> resto
1
```

Operatore modulo

- Può essere utile a verificare se un numero è **divisibile** per un altro

```
>>> 6 % 3
0
>>> 6 % 5
1
```

- Può essere usato per ricavare da un numero il valore della cifra o delle cifre più **a destra.**

```
>>> 12345 % 10
5
>>> 12345 % 100
45
>>> 1234 % 1000
234
```

Esercizio 1 - Verifica di numeri pari e dispari

Scrivi un programma Python che richieda all'utente di inserire un numero intero.

Il programma deve verificare se il numero inserito è pari o dispari utilizzando l'operatore modulo. Stampa un messaggio appropriato a seconda del risultato.

Espressioni booleane

Un'***espressione booleana*** è un'espressione che può assumere solo i valori *vero* o *falso*.

```
>>> 5 == 5
True

>>> 5 == 6
False
```

```
>>> type(True)
<class 'bool'>

>>> type(False)
<class 'bool'>
```

Operatori di comparazione

- x != y ➡ x is not equal to y
- x > y ➡ x is greater than y
- x < y ➡ x is less than y
- x >= y ➡ x is greater than or equal to y
- x <= y ➡ x is less than or equal to y
- x is y ➡ x is the same as y
- x is not y ➡ x is not the same as y

Operatori logici

Gli ***operatori logici*** in Python sono utilizzati per eseguire operazioni logiche tra valori booleani o espressioni condizionali.

and

Gli ***operatori logici*** in Python sono utilizzati per eseguire operazioni logiche tra valori booleani o espressioni condizionali.

```
>>> x = 7
>>> x > 0 and x < 10
True
```

or

Gli ***operatori logici*** in Python sono utilizzati per eseguire operazioni logiche tra valori booleani o espressioni condizionali.

```
>>> x = 8
>>> x % 2 == 0 or x % 3 == 0
True
```

not

Gli ***operatori logici*** in Python sono utilizzati per eseguire operazioni logiche tra valori booleani o espressioni condizionali.

```
>>> x = 5
>>> y = 9
>>> x > y
False
>>> not (x > y)
True
```

Esercizio 2 - Segno del prodotto

Scrivere un programma che, dati in ingresso due numeri interi non nulli qualsiasi (positivi o negativi), indichi il segno del loro prodotto senza eseguire moltiplicazioni o altre operazioni aritmetiche.

Iterazione

Aggiornamento delle variabili

Uno schema *comune* è l'aggiornamento di una variabile con un *nuovo* valore che dipende da quello *vecchio*

```
>>> x = 0
>>> x = x + 1
>>> x = x - 1
```

L’istruzione *while*

1. Il **while** valuta la condizione, comportandosi in base alla condizione
2. Se la condizione è **falsa**, esci dall’iterazione while e prosegui con l’**istruzione successiva**
3. Se la condizione è **vera**, esegui il blocco e **torna al passaggio 1**

```
contatore = 1

while contatore <= 5:
    print(contatore)
    contatore += 1

print(“Cinque!”)
```

L’istruzione *for*

L’istruzione ***for*** in Python viene utilizzata per iterare su una sequenza di elementi o su un oggetto iterabile, eseguendo un blocco di codice per ogni elemeno della sequenza o dell’oggetto.

```
friends = ['Joseph', 'Glenn', 'Sally']
for friend in friends:
    print('Happy New Year:', friend)

print('Done!')
```

Schemi di ciclo

- Inizializzazione di una o più variabili prima dell’inizio del ciclo;

- Esecuzione di alcuni calcoli su ciascun elemento nel blocco del ciclo, eventualmente modificando le variabili nel blocco del ciclo;

- Osservazione delle variabili risultanti al termine del ciclo.

Esercizio 3 - Calcolo della Media di una Lista di Voti

Scrivi un programma Python che calcoli la media di una lista di voti.

voti = [85, 92, 78, 90, 88]

Esercizio 4 - Trova il valore minimo o massimo in una lista

Scrivi un programma Python che trovi il valore minimo e massimo in una lista di numeri.

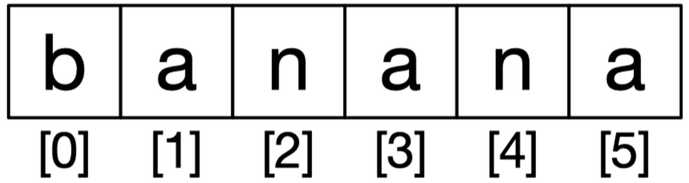
numeri = [3, 41, 12, 9, 74, 15]

Stringhe

Definizione di *stringa*

Una **stringa** è una **sequenza di caratteri *immutabile***.

È possibile accedere ad ogni carattere della stringa con l'operatore [*i*] dove *i* rappresenta l'*indice* e indica quale carattere della sequenza si vuole accedere.




```
>>> fruit = 'banana'
>>> letter = fruit[1]
'a'
```

Funzione *len*

La funzione ***len*** restituisce il numero di caratteri di una stringa

```
>>> fruit = 'banana'
>>> lunghezza = len(fruit)
>>> print(lunghezza)
6
```

Per ottenere l'ultima lettera di una stringa, l'indice giusto è *len(string) - 1* oppure utilizzare *l'indice negativo*

```
last = fruit[lunghezza - 1]
```

```
last = fruit[-1]
```

Scorrere una stringa col ciclo *for*

Ad ogni iterazione, il carattere successivo della stringa contenuta in ‘fruit’ viene assegnato alla variabile ‘*char*’.

```
for char in fruit:
    print(char)
```

Segmenti di stringhe

Il segmento di una stringa è chiamato ***slice***.

```
s = 'Walter White'
s[0:6]
‘Walter’
s[7:12]
‘White’
```

```
>>> fruit = 'banana'
>>> fruit[:3]
'ban'
>>> fruit[3:]
‘ana'
>>> fruit[3:3]
‘'
```

L'operatore *in*

La parola ***in*** è un operatore booleano che prende due stringhe e restituisce *True* se la prima è *contenuta* nella seconda:

```
>>> print('a' in 'banana')
True
>>> print('seed' in 'banana')
False
```

Esercizio 5 - Cicli e conteggi

Scrivi uno script Python che conti quante volte una specifica lettera appare in una data stringa.

string = “CyberChallenge.IT è il primo programma di addestramento in cybersecurity per studentesse e studenti universitari e delle scuole superiori ”

Metodi delle stringhe

Le stringhe in Python sono oggetti che contengono dati e **metodi incorporati**.

Invocare un metodo è simile a chiamare una funzione.

Tuttavia, non va utilizzata la sintassi *upper(word)*, ma piuttosto *word.upper()*.

string.capitalize()

Il metodo capitalize rende la prima lettera maiuscola e le successive minuscole.

```
>>> word = 'hello'
>>> cap_word = word.capitalize()
capitalized_word
Hello
```

string.upper()

Il metodo upper converte tutti i caratteri in maiuscolo.

```
>>> word = 'banana'
>>> upper_case_word = word.upper()
>>> upper_case_word
BANANA
```

string.lower()

Il metodo lower converte tutti i caratteri in minuscolo.

```
>>> word = 'BANANA'
>>> lower_case_word = word.lower()
>>> lower_case_word
banana
```

string.find()

Il metodo find cerca la posizione di una stringa o segmento in un'altra.

```
>>> word = 'banana'
>>> position = word.find('na')
>>> position
2
```

string.strip()

Il metodo strip rimuove lo spazio bianco (spazi, tabulazioni, newline) dall'inizio e dalla fine di una stringa.

```
>>> line = ' Here we go '
>>> stripped_line = line.strip()
>>> stripped_line
Here we go
```

string.split()

Il metodo split suddivide una stringa in una lista di sottostringhe in base a un delimitatore specificato.

```
>>> sentence = "Python è un linguaggio di programmazione"
>>> word_list = sentence.split()
>>> word_list
['Python', 'è', 'un', 'linguaggio', 'di', 'programmazione']
```

string.startswith()

Il metodo startswith verifica se una stringa inizia con un certo prefisso, restituendo un valore booleano.

```
>>> line = 'Have a nice day'
>>> startswith_res = line.startswith('have')
startswith_result
False
```

string.count()

Il metodo count restituisce il numero di occorrenze di una specifica sequenza o carattere all'interno di una stringa.

```
>>> line = 'Have a nice day'
>>> count_a = line.count('a')
>>> count_a
3
```

Esercizio 6 - Metodi delle stringhe

Scrivi uno script Python che, considerando il seguemte frammento di DNA =
“acactcgagacaatcttggtatcggtctacgcctcgcatcgattaggtgattgtggagcgt
cgggagtatggtatcaagcgaacttaatcctttatgtaaaggcgctttggatctttgaaga
ccagccacgtgcccgctgaccgacagctcagaacataacacgttggtcgttacccggct
aagcgaaaacgggatggggcgtcgcttcggattacccgattctgaatattcgtgtaagcat
tg cccgtacatttgtgactatatgagtaggaacgaccttgcgtccaaagaagtttagttggtt
caacgaattaacagcctagcacatagctaagtacgtcggttcatatggcccctcaccataa”

Esercizio 6 - Metodi delle stringhe

- Verifichi se contiene le seguenti sottostringhe:
  - “atcgattaggtgattgtggagcgtcggg”
  - “atcg”
- Prenda i primi 100 caratteri e li salvi in una variabile chiamata ‘sequenza’
- Trasformi tutti i caratteri in maiuscolo
- Rimuova le “a” all’inizio e alla fine
- Verifichi la lunghezza della nuova strina
- Conti le occorrenze ti timina, guanina, citosina e Adenina
- Conti quante volte compare il codone “ttg”

Liste

Definizione di *lista*

Una **lista** è una **sequenza *mutabile* di valori** chiamati ***elementi*** che possono essere di diversi tipi.

```
numbers = [10, 20, 30, 40]
animals = ['frog', 'dog', ‘horse’]
nested_list = ['spam', 2.0, 5, [10, 20]]
empty_list = []
```

Definizione di *lista*

Le liste sono ***modificabili*** in quanto è possibile modificare l'ordine degli elementi in una lista o riassegnare un elemento in una lista.

```
>>> numbers = [17, 123]
>>> numbers[1] = 5
>>> numbers
[17, 5]
```

Gli operatori ***in*** e ***not in,*** la funzione ***len***, la ***concatenazione*** e lo ***slicing*** funzionano come per le stringhe!

Scorrere una lista col ciclo *for*

Per scrivere o aggiornare gli elementi di una lista bisogna lavorare con gli indici

```
for i in range(len(numbers)):
    numbers[i] = numbers[i] * 2
```

Metodi delle liste

Anche le liste in Python sono oggetti possiedono **metodi incorporati** in grado di manipolarli.

list.append()

Il metodo append aggiunge un nuovo elemento alla fine della lista

```
>>> t = ['a', 'b', 'c']
>>> t.append('d')
>>> t
['a', 'b', 'c', 'd']
```

list.extend()

Il metodo extend accoda tutti gli elementi all’elenco specificato

```
>>> t = ['a', 'b', 'c']
>>> t.extend('d', 'e')
>>> t
['a', 'b', 'c', ‘d’, ‘e’]
```

list.sort()

Il metodo sort ordina gli elementi dell'elenco in ordine crescente

```
>>> t = ['d', 'c', 'e', 'b','a']
>>> t.sort()
>>> t
['a', 'b', 'c', 'd', 'e']
```

list.pop()

Il metodo pop rimuove e restituisce l’elemento all’indice specificato

```
>>> t = ['a', 'b', 'c']
>>> x = t.pop(1)
>>> t
‘b’
```

list.del()

Il metodo del rimuove l’elemento all’indice specificato, senza restituirlo. Si può utilizzare anche con lo slicing.

```
>>> t = ['a', 'b', 'c', 'd',
>>> 'e', 'f']
>>> del t[1:5]
>>> t
['a', 'f']
```

list.remove()

Il metodo remove rimuove l'elemento specificato

```
>>> t = ['a', 'b', 'c']
>>> t.remove('b')
>>> t
['a', 'c']
```

Liste e funzioni

Esistono numerose funzioni integrate dedicate alle liste che consentono di scorrerle rapidamente senza scrivere del codice apposito

```
>>> nums = [3, 41, 12, 9, 74, 15]
>>> max(nums)
74
>>> min(nums)
3
>>> sum(nums)
154
```

Esercizio 7 - Liste e cicli

Scrivi uno script Python che modifichi la tua lista della spesa. Hai scoperto di aver scritto il tipo di riso sbagliato, ti serve il carnaroli, non il venere! Poi hai dimenticato di aggiungere lo yogurt. Infine, hai deciso di togliere il cioccolato perché hai mangiato troppo per le feste di Natale.

shopping_list = ["pane", "latte", "cioccolato", "riso basmati", "mortadella", "parmigiano"]

Matrici

Definizione di *matrice*

Le liste annidate sono spesso usate per rappresentare ***matrici.***

```
Matrice = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Operazioni sulle matrici

Selezionare una riga

```
>>> Matrice[1]
[4, 5, 6]
```

Selezionare un elemento

```
>>> Matrice[1][1]
5
```

Esercizio 8

Durante la scansione antivirus più recente, è emerso che i nostri sistemi sono infetti da un malware di ultima generazione che si espande nella memoria. Per risolvere il problema, la memoria può essere considerata come una griglia bidimensionale con N righe e M colonne, dove ogni cella può essere di tre tipi:

- Cellula vuota (simbolo .), che non contiene dati.
- Cellula dati (simbolo +), che contiene informazioni valide.
- Cellula malware (simbolo *), che contiene parte del malware.

Esercizio 8

Ogni cella sulla griglia, dopo l'esecuzione di un software nel sistema (chiamato "round di esecuzione" per semplicità), può aggiornare il suo valore in base al numero di celle non vuote nel suo vicinato (nelle 8 direzioni possibili, o meno se è una cella di bordo), in particolare:

- Una cella vuota con più di 4 celle non vuote diventa una cella dati.
- Una cella dati con più di 4 celle non vuote diventa una cella malware.
- Una cella dati con meno di 4 celle non vuote diventa una cella vuota.
- Una cella malware con più di 4 celle non vuote diventa una cella dati.
- Una cella malware con meno di 4 celle non vuote diventa una cella vuota.

Esercizio 8

Obiettivo: Scrivere un programma che, data la configurazione iniziale di una memoria, trova la sua configurazione finale dopo *k* round.

Input: Un file che contiene:

- Nella prima riga tre interi separati da spazi: N (numero di righe della memoria), M (numero di colonne della memoria), K (numero di round da simulare).
- Le seguenti N righe contengono una stringa di M caratteri che rappresenta la configurazione iniziale della memoria

Output: N righe con una stringa di M caratteri che rappresenta la configurazione finale della memoria dopo K round.

Esercizio 8

input	output
5 4 2 +**. .+** *..+ +*+. **..	.... .+*. .*+. .+.. ....

[+**.] → [.**.] → [....]
[.+**] → [.*+.] → [.+*.]
[*..+] → [.++.] → [.*+.]
[+*+.] → [++..] → [.+..]
[**..] → [.*..] → [....]