04. Concetti sintattici fondamentali

Algoritmi e Linguaggi di Programmazione Python/C

Angelo Cardellicchio
angelo.cardellicchio@unifg.it

Outline

- Regole sintattiche fondamentali
- Programmazione strutturata in Python
- La funzione **`range()`**
- Le istruzioni **`break`** e **`continue`**
- Definire una funzione

Regole sintattiche fondamentali

- Oltre al duck typing, ci sono alcune regole fondamentali che dobbiamo ricordare quando programmiamo in Python.
- Le **parentesi**:
  - **tonde** sono obbligatorie quando chiamiamo una funzione, o se vogliamo esprimere delle precedenze nelle operazioni. In tutte gli altri casi, sono opzionali;
  - **quadre** sono usate per la creazione e l'accesso agli elementi di una struttura dati;
  - **graffe** sono usate per la creazione di un dizionario.
- L'**ambito** è delimitato mediante l'**indentazione**
- Il termine di un'istruzione è contrassegnato andando a capo

Programmazione strutturata in Python (1)

- L’istruzione **if** ha una sintassi di questo tipo.

```
if condizione:
    istruzioni()
elif altra_condizione:
    altre_istruzioni()
else:
    ultime_istruzioni()
```

Programmazione strutturata in Python (2)

- Fino alla versione 3.10, Python non offriva un supporto (diretto o indiretto) allo **switch/case**.
- A partire dalla quest'ultima versione, tale supporto è stato introdotto con il nome di **pattern matching**.

```
match comando:
    case "caso 1":
        istruzioni()
    case "altro caso":
        print("Comando sconosciuto")
```

Programmazione strutturata in Python (3)

- Il ciclo **for** prevede l'iterazione su di una *sequenza*. La sintassi è:

```
for elemento in sequenza:
    istruzioni()
```

- È estremamente importante ricordare quindi che, per usare un **for**, dobbiamo avere a disposizione una sequenza *iterabile*. Ciò rende più snello il nostro codice:

```
string = "Python"
for char in string:
    print(char)
```

Programmazione strutturata in Python (4)

- Il ciclo **while** si comporta come le controparti negli altri linguaggi di programmazione. La sintassi è:

```
while(condizione):
    istruzioni()
```

- Ad esempio:

```
i = True
while (i):
    if randint(-5, 5) > 0:
        print("Continuo!")
    else:
        print("Esco!")
        i = False
```

La funzione `range()`

- La funzione **`range()`** ci permette di definire un range di valori.
- La sintassi è del tipo:

```
range(i, j, s)
```

- Nella precedente, **`i`** è il valore iniziale (incluso), **`j`** il valore finale (escluso), ed **`s`** è il passo.
- Ad esempio, la seguente funzione restituisce tutti i numeri interi da 0 a 4:

```
r = range(0, 5, 1)
```

- È importante notare come il valore restituito non sia una lista, e vada **convertito**!

Le istruzioni **break** e **continue**

- L'istruzione **break** ci permette di uscire da un ciclo (ovvero *interromperlo*).
- L'istruzione **continue** ci permette di uscire dall'*iterazione corrente*.

```
while (True):
    if randint(-5, 5) > 0:
        print("Continuo!")
        continue
    else:
        print("Esco!")
        break
```

Definire una funzione

- Le funzioni sono delineate dalla keyword **`def`**.
- La sintassi generale è:

```
def nome_funzione(parametri):
    # istruzioni
    return valore_ritorno
```

- Notiamo che **non viene definito un tipo di ritorno, né per i parametri**.
- Se si omette l'istruzione **`return`**, si presuppone che la funzione non restituisca alcunché.
- È anche consentito inserire dei parametri con valori di **default**.
- Python prevede il passaggio esclusivamente **per valore**.

Domande?

42