Esercitazione3

October 15, 2018

1 Lezione 3

Il metodo Python **str.format()** della classe string consente di fare sostituzione di variabili e formattazione dei valori.

Questo permette di concatenare gli elementi all'interno di una stringa attraverso una formattazione posizionale.

Lo scopo del metodo **format()** è quello di rendere il codice e i programmi più leggibili ed user-friendly.

1.1 Come funziona

Un placeholder, definito attraverso una coppia di parentesi graffe {}, viene inserito in una stringa **str**.

Una volta definita la stringa, viene chiamato il metodo, cioè **str.format()**.

Quando il codice verrà eseguito, il parametro passato al metodo **format()** sarà posizionato nello stesso punto in cui è stato definito il placeholder.

```
In [75]: print("Il Real Madrid ha {} Champions League.".format(13))

Il Real Madrid ha 13 Champions League.
```

Nell'esempio precedente, abbiamo costruito una stringa con un paio di parentesi graffe come placeholder:

```
In [76]: "Il Real Madrid ha {} Champions League."

Out[76]: 'Il Real Madrid ha {} Champions League.'
```

Abbiamo quindi aggiunto il metodo **str.format()** e passato il valore intero **13** a quel metodo.

Questo colloca il valore **13** nella stringa, nella posizione in cui si trovavano le parentesi graffe:

```
In [77]: "Il Real Madrid ha 13 Champions League."

Out[77]: 'Il Real Madrid ha 13 Champions League.'
```

Possiamo anche assegnare ad una variabile il valore di una stringa in cui è stato inserito un placeholder:

```
In [78]: place_holder = "Agli studenti di ingegneria dell'informazione {}."
         print(place_holder.format("piace il linguaggio di programmazione Python!"))
```

```
Agli studenti di ingegneria dell'informazione piace il linguaggio di programmazione Python!.
```

1.2 Uso di Placeholder multipli

E' possibile usare più coppie di parentesi graffe. Per esempio:

```
In [79]: place_holder = "Agli studenti di ingegneria dell'informazione {} {}."
         print(place_holder.format("piace il linguaggio di programmazione Python!","Vero?"))
```

```
Agli studenti di ingegneria dell'informazione piace il linguaggio di programmazione Python! Ver
```

```
In [80]: place_holder = "Agli studenti di ingegneria dell'informazione {} {} {}."
         print(place_holder.format("piace il linguaggio di programmazione Python!","Vero?",";)"
```

```
Agli studenti di ingegneria dell'informazione piace il linguaggio di programmazione Python! Ver
```

1.3 Ordine e posizione degli argomenti

Quando le parentesi graffe vengono lasciate vuote ({}), Python sostituirà i valori passati attraverso il metodo **str.format()** nell'ordine in cui sono stati definiti.

```
In [81]: print("La {} è un prerequisito per {}".format("semplicità","l'affidabilità"))
```

```
La semplicità è un prerequisito per l'affidabilità
```

I due placeholder vengono riempiti tenendo conto dell'ordine dei valori contenuti nella tupla **("semplicità","l'affidabilità")**, nella posizione con indice 0 e 1, rispettivamente.

Possiamo inserire all'interno dei placeholder gli indici:

```
In [82]: print("La {0} è un prerequisito per {1}".format("semplicità","l'affidabilità"))
```

```
La semplicità è un prerequisito per l'affidabilità
```

Se noi invertiamo gli indici otteniamo:

```
In [83]: print("La {1} è un prerequisito per {0}".format("semplicità","l'affidabilità"))
```

```
La l'affidabilità è un prerequisito per semplicità
```

Se inseriamo un indice che è al di fuori del range della tupla:

```
In [84]: print("La {0} è un prerequisito per {2}".format("semplicità","l'affidabilità"))
```

```
---------------------------------------------------------------------------

IndexError                                Traceback (most recent call last)

<ipython-input-84-74df34ffca52> in <module>()
----> 1 print("La {0} è un prerequisito per {2}".format("semplicità","l'affidabilità"))


IndexError: tuple index out of range
```

```
In [ ]: print("Sammy è uno {}, {}, {} e {}!".format("squalo","blu","felice","sorridente" ))

In [ ]: print("Sammy è uno {}, {}, {} e {}!".format("squalo","blu","felice","sorridente" ))

In [ ]: print("Sammy è uno {3}, {}, {} e {}!".format("squalo","blu","felice","sorridente" ))
```

Oltre ad avere argomenti di tipo **posizionale**, è possibile introdurre delle **keyword** al posto degli indici:

```
In [ ]: print("La {0} è un prerequisito per {af}".format("semplicità", af="l'affidabilità"))

In [ ]: print("La {af} è un prerequisito per {0}".format("semplicità", af="l'affidabilità"))
```

1.4 Specificare il Tipo

Consideriamo la seguente sintassi: **{field_name:conversion}**

Dove: - **field_name** specifica l'indice dell'argomento in **str.format()**. - **conversion** fa riferimento al codice di conversione del tipo di dato che stiamo usando in format()

I codice a cui accenneremo qui sono: **s** per le stringhe, **d** per i decimali interi (base 10) ed **f** per i float. Maggiori informazioni possono essere trovare qui.

```
In [ ]: print("Pluto mangia il {0:f} percento di una {1}!".format(75,"pizza"))
```

Nell'esempio sopra c'è un numero eccessivo di numeri dopo la virgola, che possiamo limitare:

```
In [ ]: print("Pluto mangia il {0:f} percento di una {1}!".format(75.765367,"pizza"))

In [ ]: print("Pluto mangia il {0:.3f} percento di una {1}!".format(75.765367,"pizza"))

In [ ]: print("Pluto mangia il {0:.2f} percento di una {1}!".format(75.765367,"pizza"))
```

```
In [ ]: print("Pluto mangia il {0:.1f} percento di una {1}!".format(75.765367,"pizza"))
```

Notate come dimimuendo il numero di cifre dopa la virgola si otterrà un arrotondamento secondo le modalità note dell'aritmetica.

Cosa avviene se utilizziamo il tipo di conversione **d** al posto di **f**:

```
In [ ]: print("Pluto mangia il {0:d} percento di una {1}!".format(75,"pizza"))
```

Se non volessimo decimali dopo la virgola:

```
In [ ]: print("Pluto mangia il {0:.0f} percento di una {1}!".format(75.765367,"pizza"))
```

Questo non convertirà il vostro float ad un intero; non farà altro che limitare il numero di cifre dopo la virgola.

1.5 Padding delle variabili

E' possibile creare spazio intorno ad un elemento, incrementando la dimensione del campo, mediante parametri addizionali.

Questo può essere utile, ad esempio, quando è necessario organizzare una grossa quantità di dati da visualizzare.

```
In [ ]: print("Il Real Madrid ha {0:4} Champions {1:16}!".format(13,"League"))
```

Quindi, aggiungiamo un mumero per indicare la dimensione del campo (in termini di numero di caratteri), dopo i **:** all'interno delle parentesi graffe (**5** e **16**).

Di default, le stringhe sono giustificate a sinistra ed i numeri sono giustificati a destra.

E' possibile modificare questo comportamento:

- **<** allineamento testo a **sinistra**;
- ^ allineamento testo a **centro**;
- **>** allineamento testo a **destra**.

```
In [ ]: print("Il Real Madrid ha {0:<4} Champions {1:^16}!".format(13,"League"))
```

Quando utilizziamo il padding, python aggiunge degli spazi vuoti.

Possiamo modificare questo comportamento specificando il tipo di carattere da inserire:

```
In [ ]: print("{:*^20s}".format("Python"))
```

1.6 Uso di Variabili

E' possibile passare **variabili** come parametri del metodo format:

```
In [ ]: numero = 13
        print("Il Real Madrid ha {} Champions {}!".format(numero,"League"))
```

Le variabili possono essere utilizzare sia come parametri per il metodo format che per la stringa originale:

```
In [ ]: numero = 13
        team = "Il Real Madrid ha {} Champions {}!"
        print(team.format(numero,"League"))
```

1.7 Uso di format() per organizzare i dati

Un tipico uso di format() è relativo alla necessità di rendere più leggibile l'output di una qualche operazione:

```
In [ ]: for i in range(2, 12):
            print(i, i * i, i * i *i)

In [ ]: for i in range(2, 12):
            print("{:3d} {:4d} {:5d}".format(i, i * i, i * i * i))
```

Anche in questo caso possiamo manipolare l'output tenendo conto dell'allineamento (sinistra (<), centrato (^), destra (>)), oppure passare da decimale a float, etc.

1.8 Parametrizzazione

format() consente a tutte le sue componenti di essere specificate dinamicamente usando la parametrizzazione:

```
In [ ]: '{:{align}{width}}'.format('test', align='^', width='10')

In [ ]: '{:.{prec}} = {:.{prec}f}'.format('Gibberish', 2.7182, prec=3)

In [ ]: t = '{:{width}.{prec}f}'.format(2.7182, width=5, prec=2)
        print(t)
        print(type(t))

In [ ]: '{:{}{}{}.{}}'.format(2.7182818284, '>', '+', 10, 3)
```

Nome	Punteggio
Pippo	10
Pluto	8
Paperino	7

Immagine 0

1.9 f-String PEP 498

E' possibile embeddare all'interno di una stringa delle espressioni Python:

```
In [ ]: name = 'Francesco'
        f'Hello, {name}!'

In [ ]: a = 5
        b = 10
        f'Cinque più dieci è {a + b} e non {2 * (a + b)}.'
```

---

2 Esercizi

1. Stampare una stringa formattata che considera i sequenti 4 elementi:

   2, 123.4567, 10000, 12345.67 e produce 'file_002 : 123.46, 1.00e+04, 1.23e+04'

2. Stampare una tabella con la seguente forma:

3. Stampare la prima lettera, l'ultima lettera e la parola di lunghezza tre della seguente stringa:

   "Stampare la prima lettera, l'ultima lettera e la parola di lunghezza tre della seguente stringa"
   Output: La prima lettera è S, l'ultima lettera è a e la parola di lunghezza tre è tre

4. Dato n = 39 in base 10, stampare n in base 2.

5. La domenica di Pasqua è la prima domenica dopo la prima luna piena successiva all'equinozio di Primavera. Per calcolare la data precisa è possibile usare il seguente algoritmo inventato da Gauss:

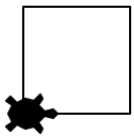
   - Sia y un qualsiasi anno (e.g. 2019)
   - Dividi y per 19 e salva il resto nella variabile a. Ignora il quoziente.
   - Dividi y per 100 e salva il quozionte nella variabile b e il resto nella variabile c.
   - Dividi il valore di b per 4 e salva il quoziente nella variabile d il resto nella variabile e.
   - Dividi 8 * b + 13 per 25 e salva il quoziente nella variabile g. Ignora il resto.

- Dividi 19 * a + b - d - g + 15 per 30 e salva il resto nella variabile h. Ignora il quoziente.
- Dividi c per 4 e salva il quoziente nella variabile j e il resto nella variabile k.
- Dividi a + 11 * h per 319 e salva il quoziente nella variabile m. Ignora il resto.
- Dividi 2 * e + 2 * j - k - h + m + 32 per 7 e salva il resto nella variabile r. Ignora il quoziente.
- Dividi h - m + r + 90 per 25 e salva il quozionte nella variabile n. Ignora il resto.
- Dividi h - m + r + n + 19 per 32 e salva il resto nella variabile p. Ignora il quozionte.

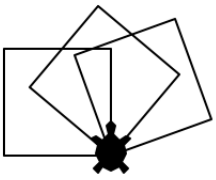
Il giorno di Pasqua cade il giorno p del mese n.

Scrivere un programma che chiede all'utente di inserire un anno (e.g. 2001) e stampa il mese e il giorno della domenica di Pasqua:

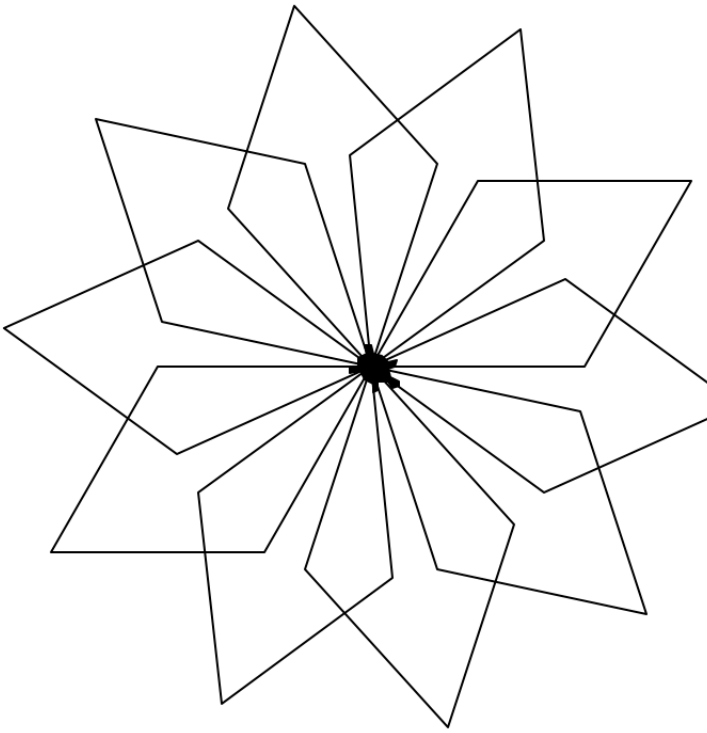
"Nell'anno 2001 la domenica di Pasqua cade il giorno 15 del mese di Aprile"



6. Usare Turtle per disegnare l'immagine in figura:



7. Usare Turtle per disegnare l'immagine in figura:



8. Usare Turtle per disegnare l'immagine in figura: