

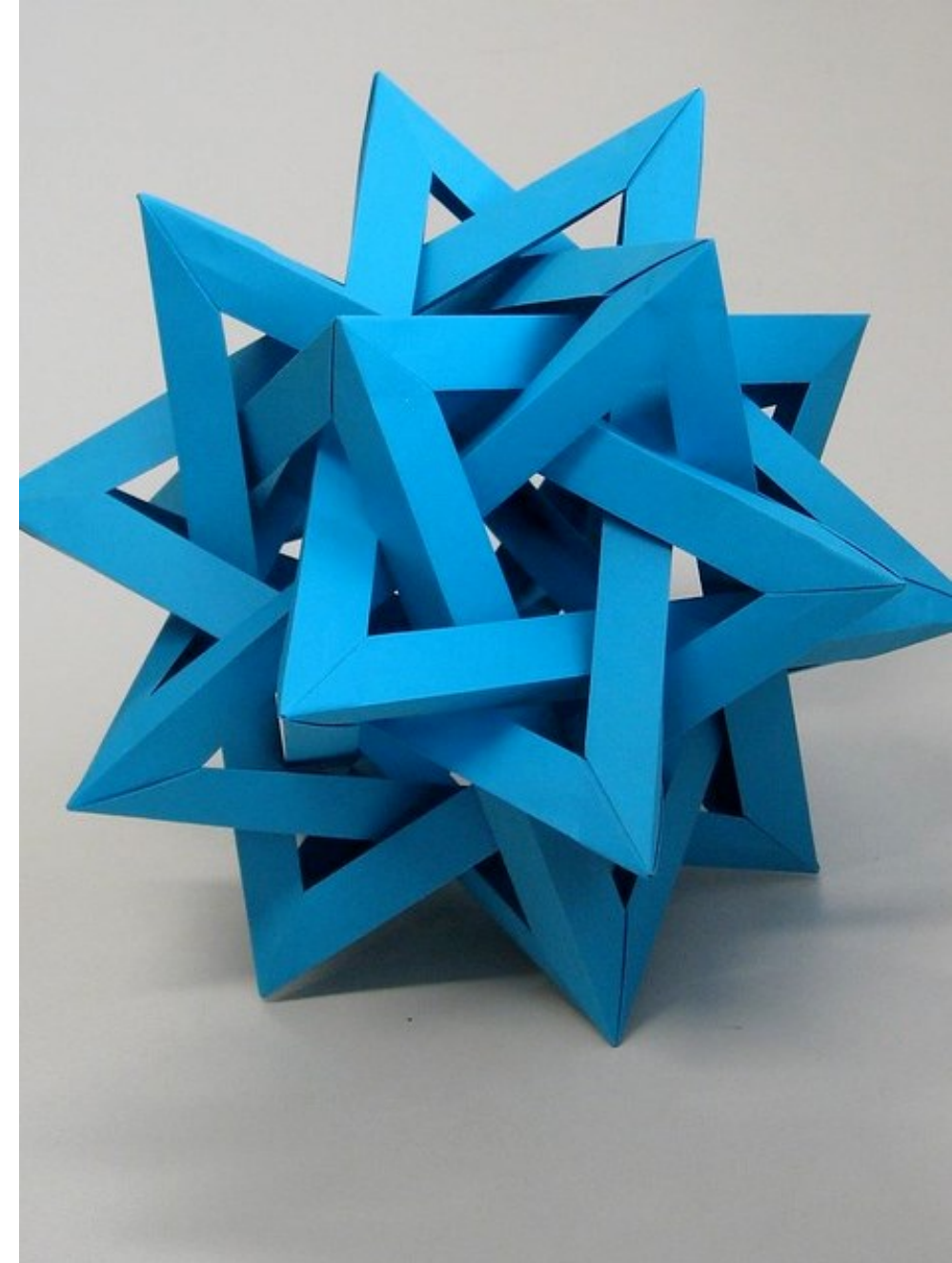


Unità P2: Numeri e stringhe

VARIABILI, VALORI, TIPI, ESPRESSIONI



Capitolo 2



Introduzione

- I **numeri** e le **stringhe** di caratteri (sequenze di caratteri) sono importanti strutture di dati in qualsiasi programma Python
 - Questi sono anche i componenti che usiamo per costruire strutture dati più complesse
- In questa Unità impareremo a utilizzare numeri e testi.
- Scriveremo diversi semplici programmi che li utilizzano.

Obiettivi dell'Unità

- **Dichiarare** e **inizializzare** variabili e costanti
- Capire le proprietà e i limiti dei **numeri interi** e a **virgola mobile**
- Apprezzare l'importanza dei **commenti** e di una buona impostazione del codice
- Scrivere espressioni matematiche ed istruzioni di assegnazione
- Creare programmi che leggano e processino input, mostrando a video i risultati
- Imparare ad usare le stringhe di Python

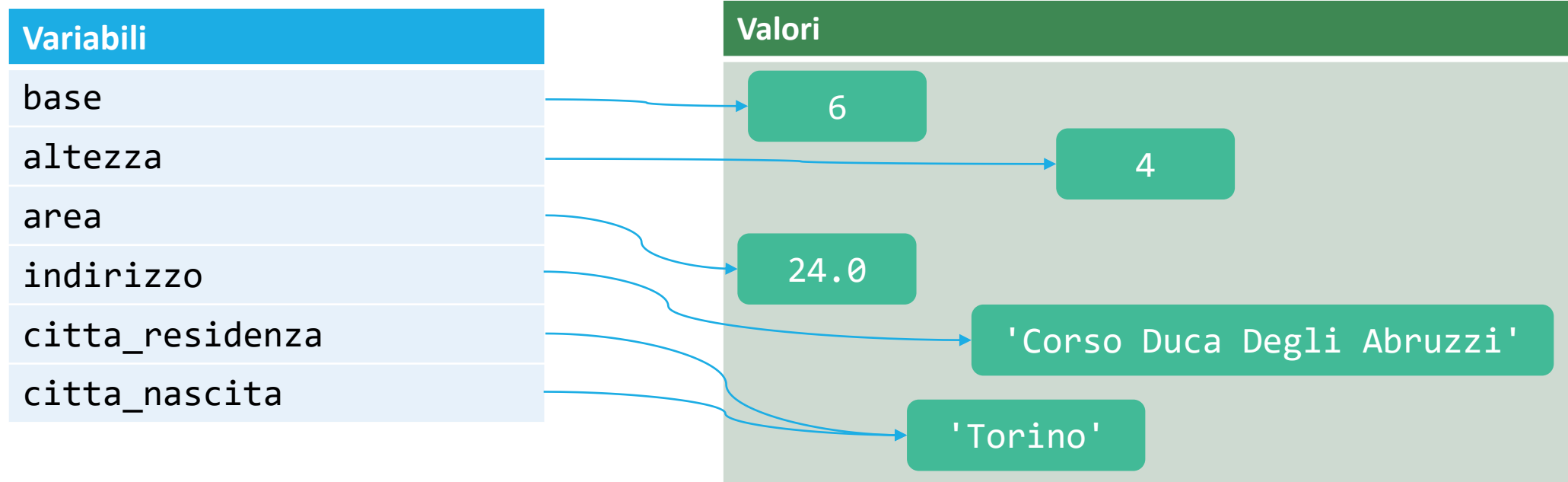
Variabili



2.1

Variabili

- Una **variabile** è una zona di memoria dotata di un nome in un programma che fa riferimento ad un **valore** specifico
- Ci sono diversi tipi di valori, ciascuno usato per memorizzare cose diverse



Variabili

- Si 'definisce' una variabile dicendo all'interprete
 - Il **nome** scelto per la variabile (con il quale, in seguito, ci si riferirà alla variabile)
 - Il **valore** iniziale della variabile

```
lattine = 4    # definisce & inizializza la variabile  
lattine
```

- Si usa un'*istruzione di assegnazione* per assegnare un valore alla variabile
 - Il valore iniziale o un nuovo valore (che sostituisce il precedente)

```
lattine = 7    # cambia il valore
```

Definizione delle variabili

- Per **definire** una variabile bisogna specificarne un **nome** e un **valore iniziale**

```
lattine = 4 # definisce & inizializza la variabile  
lattine
```

Sintassi

nomeDiVariabile = valore

Esempio

Una variabile viene definita nel momento in cui le si assegna un valore per la prima volta.

```
total = 0
```

```
...
```

```
total = bottles * BOTTLE_VOLUME
```

```
...
```

```
total = total + cans * CAN_VOLUME
```

Lo stesso nome può comparire a sinistra e a destra del segno = (Figura 2).

Nomi di variabili definite in precedenza.

Espressione che va a sostituire il valore precedente.

Nomi di variabili definite in precedenza.

Visualizzare le variabili <http://pythontutor.com/>

Write code in Python 3.6 (drag lower right corner to resize code editor)

```
1 # Constants
2
3 BOTTLE_VOLUME = 0.75
4 CAN_VOLUME = 0.33
5
6
7 # Input data
8
9 bottles = 4
10 cans = 6
11
12 # Computation
13
14 total = 0
15
16 total = bottles * BOTTLE_VOLUME
17
18 total = total + cans * CAN_VOLUME
```

Global frame

Frames	Objects
BOTTLE_VOLUME	float 0.75
CAN_VOLUME	float 0.33
bottles	int 4
cans	int 6
total	float 4.98

Valori e tipi

Nomi delle variabili

volume0.py

→ line that just executed
→ next line to execute

<< First < Prev Next > Last >>

Done running (7 steps)

L'istruzione di assegnazione

- Usare l'**istruzione di assegnazione =** per inserire un nuovo valore all'interno di una variabile: il nome della variabile si riferirà al nuovo valore
 - `lattine = 4 #` definisce `lattine` e inizializza a 4
 - `lattine = 6 #` **cambia il valore** associato a `lattine`, che da ora sarà 6
- Attenzione: il segno **= NON** rappresenta un confronto:
 - Esso copia il valore alla destra del segno nella variabile alla sinistra del segno
 - Vedrete l'operatore di verifica di uguaglianza nel prossimo capitolo

Sintassi dell'assegnazione

- Il valore calcolato alla destra del segno '=' è assegnato alla variabile sulla sinistra

Sintassi

nomeDiVariabile = valore

Esempio

Una variabile viene definita nel momento in cui le si assegna un valore per la prima volta.

total = 0

...

total = bottles * BOTTLE_VOLUME

...

total = total + cans * CAN_VOLUME

Lo stesso nome può comparire a sinistra e a destra del segno = (Figura 2).

Nomi di variabili definite in precedenza.

Espressione che va a sostituire il valore precedente.

Nomi di variabili definite in precedenza.

Un esempio: acquisto di una soda

- Le bevande analcoliche sono vendute in lattina e in bottiglia. Un negozio ne offre una confezione di 6 lattine da 12 once allo stesso prezzo una bottiglia da due litri. Quale conviene comprare? (12 once fluide equivalgono a circa 0.355 litri)

Lista delle variabili:

Numero di lattine a confezione
Once per lattina
Once per bottiglia

Tipi di numeri:

Numero intero
Numero intero
Numero con frazione

Perché diversi tipi?

- Abbiamo visto tre diversi tipi di dato:
 - Numero **intero** (senza parte frazionaria) 7 (**integer** or int)
 - Numero con parte **frazionaria** 8.88 (**float**)
 - Sequenza di **caratteri** "Bob" (**string**)

- Il **tipo** è associato al valore, non alla **variabile**
 - `cansPerPack = 6 # int`
 - `canVolume = 12.0 # float`

Aggiornare una variabile (assegnare un valore)

- Se un nuovo valore viene associato a una variabile esistente, quel valore rimpiazza il contenuto precedente della variabile

- Per esempio:

- cansPerPack = 6

① ②

- cansPerPack = 8

③

① Trattandosi della prima assegnazione, la variabile viene creata.

cansPerPack =

② La variabile viene inizializzata.

cansPerPack =

③ La seconda assegnazione sovrascrive il valore memorizzato.

cansPerPack =

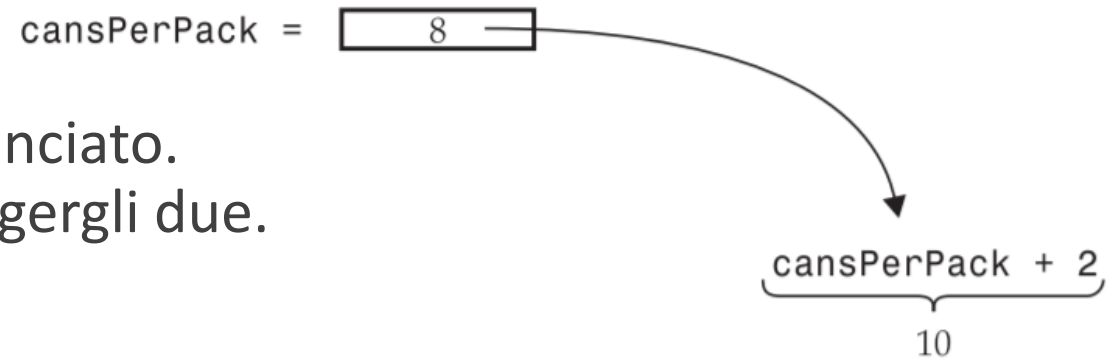
Aggiornare una variabile (calcolata)

- Esecuzione dell'enunciato:
 - $\text{cansPerPack} = \text{cansPerPack} + 2$

- Step by Step:

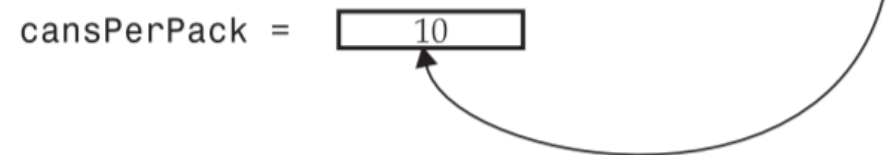
- Step 1: **Calcolare** la parte **destra** dell'enunciato.
Trovare il valore di `cansPerPack`, e aggiungergli due.

① Calcola il valore della parte destra



- Step 2: **Salvare il risultato** nella variabile a sinistra dell'operatore di assegnazione

② Memorizza il valore nella variabile



Variabili non definite

- Bisogna **definire** (e inizializzare) una variabile **prima** di usarla: deve essere definita da qualche parte sopra la linea di codice dove la si usa per la prima volta
 - `canVolume = 12 * literPerOunce`
 - `literPerOunce = 0.0296`
 - Will cause a **NameError: name 'literPerOunce' is not defined**
- L'ordine corretto delle istruzioni è:
 - `literPerOunce = 0.0296`
 - `canVolume = 12 * literPerOunce`

Tipi di dati

- Il **tipo** di dato è associato al valore e non alla **variabile**. Quindi:
- Una variabile può essere assegnata a **diversi valori di diverso tipo**, in diverse parti del programma.

- `taxRate = 5` # un intero

E poi...

- `taxRate = 5.5` # un numero a virgola mobile

E poi...

- `taxRate = "Non-taxable"` # una stringa

- Se si usa una variabile ed essa è di un tipo non previsto, il programma darà errore.

Esempio...

- Aprire PyCharm (o replit.com) e creare un nuovo progetto
- # Testing different types in the same variable
taxRate = 5 # int
print(taxRate)
taxRate = 5.5 # float
print(taxRate)
taxRate = "Non-taxable" # string
print(taxRate)

Attenzione!

- Il seguente programma stampa `5` , `5` , `Non-Taxable`. Perché? Qual è l'errore?

- `# Testing different types in the same variable`
`taxRate = 5 # int`
`print(taxRate)`
`taxrate = 5.5 # float`
`print(taxRate)`
`taxRate = "Non-taxable" # string`
`print(taxRate)`

Ma...

- `taxRate = "Non-taxable" # string`
`print(taxRate)`
`print(taxRate + 5)`
 - Genera: **TypeError**: can only concatenate str (not "int") to str
- `print(taxRate + "??")`
 - Stampa Non-taxable??
- Quindi...
 - Una volta inizializzata una variabile con un valore di un particolare tipo si deve avere cura di continuare a salvare nella variabile valori del medesimo tipo



Avvertimenti

- Vanno eseguite solamente le operazioni che sono valide a seconda del valore corrente della variabile
 - Ricordarsi a che tipo si riferisce ogni variabile
- Quando si utilizza l'operatore “+” con stringhe, il secondo argomento viene concatenato alla fine del primo, ma entrambi gli argomenti devono essere stringhe
 - Le operazioni sulle stringhe verranno viste più nel dettaglio più avanti in questa Unità

Tabella 1: Numeri espliciti (*literal*) in Python

Tabella 1

Numeri letterali in Python






Numero	Tipo	Commento
6	int	Un numero intero non ha parte frazionaria.
-6	int	I numeri interi possono essere negativi.
0	int	Zero è un numero intero.
0.5	float	Un numero con parte frazionaria è di tipo float.
1.0	float	Un numero intero con parte frazionaria <code>.0</code> è di tipo float.
1E6	float	Un numero in notazione esponenziale: 1×10^6 , cioè 1 000 000. I numeri in notazione esponenziale sono sempre di tipo float.
2.96E-2	float	Esponente negativo: $2.96 \times 10^{-2} = 2.96 / 100 = 0.0296$.
 100,000		Errore: non usare la virgola (né il punto) come separatore delle migliaia.
 3 1/2		Errore: non usare frazioni, soltanto la notazione decimale (qui, 3.5).

Nomi delle variabili

- Seguire queste semplici regole
 - I nomi delle variabili devono iniziare con una lettera o con il carattere underscore-sottolineato (`_`)
 - I caratteri seguenti possono essere lettere (maiuscole o minuscole), numeri o underscore
 - Non si possono usare altri simboli (`?` or `%`, ...), né gli spazi
 - Separare le parole secondo la convenzione
 - `'snake_case'`: Utilizzare gli underscore per separare le diverse parole (preferito)
 - `'camelCase'`: Utilizzare le maiuscole per indicare i confini delle diverse parole
 - Non utilizzare le parole `'riservate'` di Python (vedere Appendice C, pag. A6 e A7)

Tabella 2: Nomi di variabili in Python

Tabella 2
Nomi di variabili in Python

Nome della variabile	Commento
canVolume1	I nomi delle variabili sono costituiti da lettere, da cifre e dal segno di sottolineatura.
x	In matematica si usano nomi di variabili brevi, come x o y . Anche in Python è lecito fare così, ma è una pratica poco comune, perché rende poco comprensibili i programmi (Suggerimenti per la programmazione 2.1).
 CanVolume	Attenzione: nei nomi di variabile, maiuscole e minuscole sono diverse. Questo nome di variabile è diverso da <code>canVolume</code> e viola la convenzione che prevede di iniziare i nomi di variabile con una lettera minuscola.
 6pack	Errore: un nome di variabile non può iniziare con una cifra.
 can volume	Errore: un nome di variabile non può contenere spazi.
 class	Errore: non si può usare una parola riservata del linguaggio come nome di variabile.
 ltr/fl.oz	Errore: non si possono usare simboli, come / o .

Suggerimento: nomi descrittivi

- Scegliere per le variabili dei nomi descrittivi
- Quale nome è più descrittivo?
 - `canVolume` = 0.35
 - `cv` = 0.355
- Questo risulta particolarmente importante quando un programma è scritto da più persone.

Costanti

- In Python una **costante** è una variabile il cui valore *non andrebbe modificato* dopo che le è stato assegnato un *valore iniziale*.
- È buona norma usare le **maiuscole** per nominare le costanti
 - `BOTTLE_VOLUME = 2.0`
- Come stile di programmazione si ritiene utile usare, nei calcoli, costanti dotate di nome al posto di valori espliciti: quale è più chiaro?
 - `totalVolume = bottles * 2`
 - `totalVolume = bottles * BOTTLE_VOLUME`
- Un programmatore, leggendo la prima espressione, potrebbe non capire il significato del “2”
- Python permette di modificare il valore di una costante
 - Solo perché si può fare, non significa che si deve fare

Costanti: Nomi & Stile

- È consuetudine usare le MAIUSCOLE per le costanti in modo da distinguerle dalle variabili.
 - È molto chiaro visivamente
- `BOTTLE_VOLUME = 2` `# Constant`
- `MAX_SIZE = 100` `# Constant`
- `taxRate = 5` `# Variable`

Python & Commenti

- È bene utilizzare commenti all'inizio di ogni programma e chiarire i dettagli del codice
- I commenti sono d'aiuto agli altri e un modo per tenere traccia del ragionamento
 - Commentare serve ad aggiungere spiegazioni per chi legge il codice
- Il compilatore ignora i commenti
 - Altri programmatori li leggeranno
 - Anche tu, un giorno

Documentation is a love letter that you write to your future self. Damian Conway (2005). "Perl Best Practices", p.153, "O'Reilly Media, Inc."

Commentare il codice



```
1  ##
2  # Questo programma calcola il volume (in litri) di una confezione
3  # di bibite con sei lattine, seguito dal volume di una tale confezione
4  # insieme a una bottiglia da due litri.
5
6  # Litri in una lattina da 12 once e in una bottiglia da due litri.
7  CAN_VOLUME = 0.355
8  BOTTLE_VOLUME = 2
9
10 # Numero di lattine in una confezione.
11 cansPerPack = 6
12
13 # Calcoliamo il volume totale nelle lattine di una confezione.
14 totalVolume = cansPerPack * CAN_VOLUME
15 print("A six-pack of 12-ounce cans contains", totalVolume, "liters.")
16
17 # Calcoliamo il volume totale di una confezione e di una bottiglia.
18 totalVolume = totalVolume + BOTTLE_VOLUME
19 print("A six-pack and a two-liter bottle contain", totalVolume, "liters.")
```

 volume1.py

Aritmetica



2.2

Operatori aritmetici elementari

- Python supporta tutte le operazioni aritmetiche elementari:

- Addizione `+`
- Sottrazione `-`
- Moltiplicazione `*`
- Divisione `/`
- Potenza `**`

- Usare le parentesi per scrivere le espressioni

$$\frac{a + b}{2} \rightarrow (a + b) / 2$$

$$b \times \left(1 + \frac{r}{100}\right)^n \rightarrow b * ((1 + r / 100) ** n)$$

- La precedenza è simile a quella algebrica:

- PEMDAS
 - Parenthesis, Exponent, Multiply/Divide, Add/Subtract

Utilizzare diversi tipi numerici

- Se si usano **numeri interi** e **a virgola mobile** in un'espressione matematica, il risultato sarà un **numero a virgola mobile**.
- `7 + 4.0` # Porta al valore a virgola mobile
`11.0`
- `4` e `4.0` sono tipi di dato diversi, per un computer
- Ricordare:
 - Se si usano stringhe con numeri interi o a virgola mobile, il risultato sarà un errore

Divisione intera

- Quando si dividono due numeri interi con l'operatore `/`, si ottiene un valore a virgola mobile.
 - Ad esempio, `7 / 4` dà `1.75`
- Si può però anche eseguire una **divisione intera** usando l'operatore `//`.
 - L'operatore `"//"` calcola il quoziente e ignora la parte frazionaria
 - Per esempio, `7 // 4` dà come risultato `1`
 - Infatti, 7 diviso 4 è 1.75 con una parte frazionaria di 0.75, che viene ignorata.
 - Se gli operandi sono frazionari, effettua la divisione e poi tronca il risultato

Calcolare il resto

- Se si è interessati al **resto** della divisione tra interi, va usato l'operatore “%” (detto **modulo**):
 - $\text{resto} = 7 \% 4$
- Il valore del resto sarà 3
- Talvolta detto «modulo»
- Usata prevalentemente tra numeri interi
 - Per operandi frazionari, l'operazione non è particolarmente utile

Esempio


```
# Convert pennies to dollars and cents
pennies = 1729

dollars = pennies // 100 # Calculates the number
of dollars

cents = pennies % 100    # Calculates the number
of pennies

print("I have", dollars, "and", cents, "cents")
```



 pennies.py

Esempi di divisione intera e resto

Tabella 3
Divisione intera e resto

Espressione (con $n = 1729$)	Valore	Commento
$n \% 10$	9	Per qualsiasi numero intero positivo n , $n \% 10$ è l'ultima cifra di n .
$n // 10$	172	Questo è n senza la sua ultima cifra.
$n \% 100$	29	Le ultime due cifre di n .
$n \% 2$	1	$n \% 2$ vale 0 se n è pari, vale 1 se n è dispari (con n non negativo)
$-n // 10$	-173	-173 è il più grande numero intero che sia ≤ -172.9 (ma in questo libro non useremo la divisione intera con numeri negativi)

Invocare funzioni

- Una **funzione** è un gruppo di istruzioni di programmazione che esegue un determinato compito.
- La funzione `print()` visualizza informazioni, ma **ci sono molte altre funzioni disponibili** in Python
 - Imparerete a chiamare le funzioni disponibili e a crearne delle nuove
- Invocando una funzione bisogna passare il numero corretto di **argomenti** (anche chiamati **parametri**)
 - Se non viene fatto, il programma genera un messaggio di errore

Invocare funzioni che restituiscono un valore

- **La maggior parte delle funzioni restituisce un valore.** Ossia quando una funzione completa il suo compito passa il valore al punto dove la funzione è stata invocata
- Per esempio:
 - L'invocazione `abs(-173)` restituisce il valore 173.
 - Il valore **restituito** da una funzione può essere **usato in un'espressione** o **salvato in una variabile**
 - `distance = abs(x)`
 - Si può usare l'invocazione di una funzione come argomento della funzione `print`
 - `print(abs(-173))`

Funzioni matematiche predefinite

Tabella 4
Funzioni matematiche
predefinite

Funzione	Restituisce
<code>abs(<i>x</i>)</code>	Il valore assoluto di <i>x</i> .
<code>round(<i>x</i>)</code> <code>round(<i>x</i>, <i>n</i>)</code>	Il valore <i>x</i> in virgola mobile arrotondato in modo che sia un numero intero o che abbia <i>n</i> cifre decimali.
<code>max(<i>x</i>₁, <i>x</i>₂, ..., <i>x</i>_{<i>n</i>})</code>	Il valore maggiore tra quelli presenti come argomenti.
<code>min(<i>x</i>₁, <i>x</i>₂, ..., <i>x</i>_{<i>n</i>})</code>	Il valore minore tra quelli presenti come argomenti.

Librerie di Python (moduli)

- Una **libreria** è una raccolta di codice (e.g., funzioni, costanti, tipi di dato), scritto e compilato da terzi, che è pronto all'uso in un programma
 - Sempre controllarne la documentazione prima dell'utilizzo
- Una **libreria standard** è una libreria che si considera parte del linguaggio ed è **inclusa** in qualsiasi ambiente di sviluppo Python
 - <https://docs.python.org/3/library/index.html>
- Le librerie (comprese quella standard) sono organizzate in **moduli**
 - Funzioni e tipi di dati correlati sono raggruppati nello stesso modulo
 - Le funzioni definite in un modulo devono essere esplicitamente caricate in un programma prima che questo le possa utilizzare

Librerie, moduli, funzioni

- Funzioni predefinite
 - Sempre disponibili (come `print()`, `abs()`, ...)
- Libreria standard
 - Parte di ogni installazione Python
 - Organizzate in *moduli*
 - Ogni modulo contiene più funzioni
 - Prima di usare la funzione, bisogna *importare il modulo*
- Librerie aggiuntive
 - Non fanno parte di Python
 - Devono essere scaricate/installate nel progetto (usando l'IDE o la linea di comando)
 - Dopo essere state installate, possono essere importate e se ne possono usare le funzioni

Funzioni predefinite (sempre disponibili)

Built-in Functions				
<code>abs()</code>	<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>
<code>all()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>any()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>ascii()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>bin()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bool()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>breakpoint()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	

<https://docs.python.org/3/library/functions.html>

Funzioni della libreria standard

- La funzione `sqrt()`, che calcola la radice quadrata dell'argomento, è nel modulo 'math' della libreria standard
 - <https://docs.python.org/3/library/math.html>

```
# First include this statement at the top of your  
# program file.
```

```
from math import sqrt
```

```
# Then you can simply call the function as  
y = sqrt(x)
```

Alcune funzioni dal modulo math

Tabella 5
Alcune funzioni
matematiche del modulo
math

Funzione	Restituisce
<code>sqrt(x)</code>	La radice quadrata di x (con $x \geq 0$).
<code>trunc(x)</code>	Tronca il valore x in virgola mobile, restituendo un numero intero.
<code>cos(x)</code>	Il coseno di x , in radianti.
<code>sin(x)</code>	Il seno di x , in radianti.
<code>tan(x)</code>	La tangente di x , in radianti.
<code>exp(x)</code>	e^x
<code>degrees(x)</code>	Converte x da radianti a gradi (cioè restituisce $x \cdot 180/\pi$).
<code>radians(x)</code>	Converte x da gradi a radianti (cioè restituisce $x \cdot \pi/180$).
<code>log(x)</code>	Il logaritmo naturale di x (cioè il logaritmo in base e) oppure il logaritmo di
<code>log(x, base)</code>	x nella <i>base</i> indicata.

```
from math import xxxxx
```

Importare i moduli

- Tre modi per importare le funzioni dai moduli:
 - `from math import sqrt, sin, cos`
imports listed functions
 - `from math import *`
imports all functions from the module
 - `import math`
imports the module and gives access to all functions
- Se si usa il terzo metodo, bisogna aggiungere il nome del modulo e un “.” prima di ogni invocazione di funzione
 - `import math`
 - `y = math.sqrt(x)`

Conversione da numero a virgola mobile a intero

- Si possono usare le funzioni `int()` e `float()` per convertire tra numeri interi e a virgola mobile:

```
balance = total + tax    # balance: float
```

```
dollars = int(balance)  # dollars: integer
```

- Si perde la parte frazionaria del numero a virgola mobile (non c'è arrotondamento)

Conversione da virgola mobile a intero

Function	Definition
<code>math.floor(x)</code>	Restituisce l'approssimazione per difetto di x come numero in virgola mobile.
<code>math.ceil(x)</code>	Restituisce l'approssimazione per eccesso di x come numero in virgola mobile.
<code>math.trunc(x)</code>	Restituisce il valore reale di x troncato ad un intero (<i>non</i> è equivalente a <code>floor</code> , per i numeri negativi)
<code>round(x, d)</code>	Restituisce x approssimato con una precision di d cifre dopo il punto decimale. Se d è omesso o è <code>None</code> , ritorna l'intero più vicino.
<code>int(x)</code> <code>int(s)</code>	Ritorna un valore intero a partire da un numero x o una stringa s . Se x è un numero in virgola mobile, tronca verso lo zero. Per le stringhe, converte la stringa in un intero (se la stringa rappresenta un numero).

Per maggiori informazioni:
How to Round Numbers in Python
<https://realpython.com/python-rounding>

Errori di arrotondamento

- I numeri a virgola mobile non sono esatti
 - Questa è una limitazione del sistema binario, non tutti i numeri a virgola mobile hanno una rappresentazione esatta
- Provare:

```
price = 4.35
quantity = 100
total = price * quantity
# Should be 100 * 4.35 = 435.00
print(total) # prints 434.999999999999994
```
- Si possono gestire gli errori di arrotondamento:
 - Approssimando all'intero più vicino (vedere Sezione 2.2.4)
 - Mostrando un numero fisso di numeri dopo la virgola (vedere Sezione 2.5.3)
 - Definire una tolleranza `EPSILON` e fare solo confronti approssimati (vedere Sezione 3.2, e vedere `math.isclose()`)

```
>>> 0.1+0.2
0.30000000000000004
>>> 0.01+0.02
0.03
```



Espressioni aritmetiche

Tabella 6
Esempi di espressioni aritmetiche

Espressione matematica	Espressione in Python	Commento
$\frac{x + y}{2}$	<code>(x + y) / 2</code>	Le parentesi sono necessarie, perché <code>x + y/2</code> calcolerebbe <code>x + (y/2)</code> .
$\frac{xy}{2}$	<code>x * y / 2</code>	Le parentesi non sono necessarie, perché gli operatori che hanno la stessa precedenza vengono valutati da sinistra a destra.
$\left(1 + \frac{r}{100}\right)^n$	<code>(1 + r / 100) ** n</code>	Le parentesi sono necessarie.
$\sqrt{a^2 + b^2}$	<code>sqrt(a ** 2 + b ** 2)</code>	Bisogna importare la funzione <code>sqrt</code> dal modulo <code>math</code> .
π	<code>pi</code>	La costante <code>pi</code> è dichiarata nel modulo <code>math</code> .

Parentesi non accoppiate

- Considerare l'espressione

$$((a + b) * t / 2 * (1 - t))$$

- Che cosa c'è di sbagliato?

- Ora si consideri questa espressione

$$(a + b) * t) / (2 * (1 - t))$$

- Questa espressione ha tre "(" e tre ")", ma non è ancora corretta

- In qualsiasi punto interno ad un'espressione, il numero di parentesi che sono state aperte dall'inizio dell'espressione deve essere maggiore o uguale a quello delle parentesi che sono state chiuse
- Al termine dell'espressione i due conteggi devono equivalersi

Suggerimenti per la programmazione

- Utilizzare gli spazi nelle espressioni

```
totalCans = fullCans + emptyCans
```

- È più leggibile di:

```
totalCans=fullCans+emptyCans
```

Esercizi di problem solving



2.3

PRIMA SI PENSA ALL'ALGORITMO, POI SI SCRIVE IL CODICE PYTHON

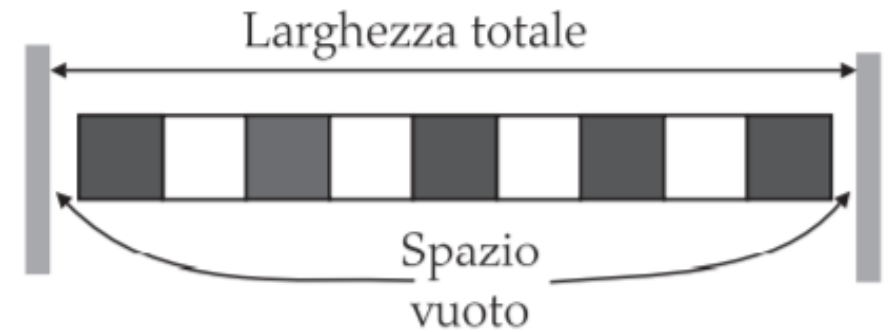
Problem solving: prima lo si fa a mano

- Un passo importantissimo dello sviluppo di un algoritmo è eseguire «a mano» l'elaborazione prevista.
 - Se non si è in grado di risolverlo a mano, difficilmente si riuscirà a scrivere il programma
- Esempio di problema:
 - Bisogna posizionare lungo il muro una fila di piastrelle nere e bianche. Per ragioni estetiche l'architetto ha specificato che la prima e l'ultima piastrella devono essere **nere**. Il vostro compito è calcolare il **numero di piastrelle** necessarie e il **vuoto** a ciascuna delle due estremità della riga, dato lo **spazio** disponibile e la **larghezza** di ogni piastrella.



Iniziare con valori d'esempio

- Dati
 - Larghezza totale: 100 pollici
 - Larghezza di una piastrella: 5 pollici
- Testare i valori
- Vediamo... $100/5 = 20$, perfetto! 20 piastrelle. Niente spazio vuoto.
 - Però... BN...BN «... la prima e l'ultima piastrella dovrebbero essere nere.»
- Bisogna guardare con più attenzione al problema...
 - Iniziando con una nera, poi con coppie BN



- Osservazione: ogni coppia è 2 volte la larghezza di 1 piastrella
 - In questo esempio, $2 \times 5 = 10$ pollici

Continuare ad applicare la soluzione

- Larghezza totale: 100 pollici
- Larghezza di una piastrella: 5 pollici



- Calcolare la larghezza totale di tutte le piastrelle
 - Una piastrella nera: 5 pollici
 - 9 coppie di BN: 90 pollici
 - Larghezza totale: 95 pollici
- Calcolare gli spazi (uno a ciascuna estremità)
 - $100 - 95 = 5$ pollici = spazio totale
 - Spazio di 5 pollici / 2 = 2.5 pollici a ciascuna estremità

Traduzione in algoritmo

- Usare l'esempio per capire come vadano eseguiti i calcoli
- Quante coppie di piastrelle B e N?
 - Notare: deve essere un numero intero
 - Parte intera di: $(\text{larghezza totale} - \text{larghezza piastrella}) / (2 * \text{larghezza piastrella})$
- Quante piastrelle?
 - $1 + 2 \times \text{numero di coppie}$
- Spazio a ciascuna estremità
 - $(\text{larghezza totale} - \text{numero di piastrelle} * \text{larghezza piastrella}) / 2$

L'algoritmo

- Calcolare il numero di coppie di piastrelle
 - Numero di coppie = parte intera di $(\text{larghezza totale} - \text{larghezza piastrella}) / (2 * \text{larghezza piastrella})$
- Calcolare il numero di piastrelle
 - Numero di piastrelle = $1 + (2 * \text{numero di coppie})$
- Calcolare lo spazio
 - Spazio ad ogni estremità = $(\text{larghezza totale} - \text{numero di piastrelle} * \text{larghezza piastrella}) / 2$
- Visualizzare il numero di coppie di piastrelle
- Visualizzare il numero di piastrelle totale in una riga
- Visualizzare lo spazio vuoto

Soluzione



```
##  
# Computes the number of tiles needed and the gap at each end when  
# placing tiles along a wall.  
#  
  
# Define the dimensions.  
totalWidth = 100  
tileWidth = 5  
  
# Calculate the tiles and gaps.  
numberOfPairs = (totalWidth - tileWidth) // (2 * tileWidth)  
numberOfTiles = 1 + 2 * numberOfPairs  
gap = (totalWidth - numberOfTiles * tileWidth) / 2.0  
  
print("Number of tiles:", numberOfTiles)  
print("Gap at each end:", gap)
```

 tiles.py

Stringhe



2.4

Stringhe

- Definizione base:
 - Un **testo** è costituito da **caratteri**
 - I **caratteri** sono lettere, numeri, segni punteggiatura, spazi, ...
 - Una **stringa** è una **sequenza di caratteri**
- In Python, le stringhe esplicite (*literal*) sono specificate racchiudendo la sequenza di caratteri tra virgolette singole o doppie.

```
print("This is a string.", 'So is this.')
```
- Permettendo entrambi i tipi di virgolette, Python rende più semplice l'inclusione di un apostrofo o di virgolette all'interno della stringa
 - `message = 'He said "Hello"'`
 - Ricordare di usare lo stesso tipo di virgolette, singole con singole e doppie con doppie

Concatenazione di stringhe (“+”)

- Si può ‘aggiungere’ una stringa alla fine di un’altra

```
firstName = "Harry"
```

```
lastName = "Morgan"
```

```
name = firstName + lastName # HarryMorgan
```

```
print("my name is:", name)
```

- Si vuole aggiungere uno spazio tra le due stringhe?

```
name = firstName + " " + lastName # Harry Morgan
```

Lunghezza delle stringhe

- Il numero di caratteri in una stringa è detto **lunghezza** della stringa.
 - Esempio, la lunghezza di "Harry" è 5
- Si può calcolare la lunghezza di una stringa usando la funzione Python `len()`:
`length = len("World!") # length è 6`
- La stringa di lunghezza **0** è detta *stringa vuota*. Non contiene caratteri ed è scritta come `""` oppure `' '`.

Note

- Usare “+” per concatenare le stringhe è un esempio di un concetto detto **operator overloading** (ossia operatori che hanno diverse funzioni a seconda dell’utilizzo).
- L’operatore “+” svolge **diverse** funzioni, dipendendo dal **tipo** di **valori** coinvolti:
 - intero + intero → addizione intera
 - float (virgola mobile) + float, float + intero → addizione float
 - stringa + stringa → concatenazione di stringhe
 - lista + lista → concatenazione di liste
 - ...
- Ma...
 - stringa + numero intero → errore

Ripetizione di stringhe (“*”)

- Si può anche produrre una stringa che sia il risultato della ripetizione di una stringa
- Ipotesizzare che si voglia visualizzare una linea tratteggiata
- Invece di specificare una stringa con 50 trattini, si può usare l’operatore `*` per creare una stringa che sia composta dalla stringa `" - "` ripetuta 50 volte
 - `dashes = " - " * 50`
 - Dà come risultato la stringa
"-----"
- Anche l’operatore `"*"` è un tipo di operatore *overloaded*

Convertire i numeri in stringhe

- Si usa la funzione **str()** per convertire da numeri a stringhe

```
balance = 888.88
```

```
dollars = 888
```

```
balanceAsString = str(balance)
```

```
dollarsAsString = str(dollars)
```

```
print(balanceAsString)
```

```
print(dollarsAsString)
```


Convertire le stringhe in numeri

- Quando una stringa contiene la rappresentazione di un numero (intero o floating point), la si può convertire in un valore numerico usando le funzioni **int()** e **float()**:

```
val = int("1729")  
price = float("17.29")  
print(val)  
print(price)
```

- Questa conversione è **importante** quando le stringhe sono fornite in input dall'utente (vedremo la prossima settimana come)

Errori di conversione

- Nel convertire una stringa in numero, è necessario che la stringa contenga una **rappresentazione corretta** del numero stesso
- In caso contrario, il programma genera un errore di tipo «Value Error»

```
val = int("ciao")
```

```
ValueError: invalid literal for int() with base 10: 'ciao'
```

Stringhe e caratteri

- Le stringhe sono sequenze di **caratteri**
- Le stringhe sono **immutabili**: non possono essere modificate dopo la loro creazione
 - La stessa variabile può però essere aggiornata per riferirla ad un'altra stringa

Estrarre un carattere da una stringa

- Ad ogni carattere in una stringa corrisponde un indice numerico

0	1	2	3	4	5	6	7	8	9
c	h	a	r	s		h	e	r	e

← indice

← carattere

- Il primo carattere ha indice zero (0)
 - L'ultimo carattere ha indice $\text{len}(\text{name}) - 1$
- L'operatore `[]` ritorna il carattere corrispondente ad un dato indice:

```
name = "Harry"  
first = name[0]  
last = name[4]
```

0	1	2	3	4
H	a	r	r	y

↑ name[0]

↑ name[4]

Indici validi

- Gli unici indici validi vanno da 0 a `len(string) - 1`
- Indici non validi generano errore
 - **IndexError: string index out of range.**

```
name = 'Bob'  
print(name, 'has length', len(name))  
print(name[0])  
print(name[1])  
print(name[2])  
print(name[3])
```

```
Bob has length 3  
B  
o  
b  
Traceback (most recent call last):  
  File "main.py", line 6, in <module>  
    print(name[3])  
IndexError: string index out of range
```

Immutabilità

- Le stringhe sono immutabili in Python
- Non si possono cambiare i valori dei caratteri
 - Viene generato l'errore **TypeError**

```
name = 'Bob'  
  
print(name[0])  
  
name[0] = 'G'
```

```
B  
Traceback (most recent call last):  
  File "main.py", line 5, in <module>  
    name[0] = 'G'  
TypeError: 'str' object does not support  
item assignment
```

- **Workaround**: impareremo a costruire stringhe 'aggiornate' invece di modificare le stringhe correnti

Porzioni (*slice*) di una stringa

- Serve ad estrarre una parte della stringa
- Data una stringa:
`nome = "Paperon de' Paperoni"`
- Si è interessati solo alla porzione di stringa dal quinto carattere (indice: 4, 'r') al decimo carattere (indice: 9, 'e')
- Si può estrarre una porzione di stringa ottenere **l'operatore di slice**:
`porzione = nome[4 : 10]`
- Gli argomenti sono il primo elemento (incluso) e l'ultimo (escluso)
 - Quindi nell'esempio si otterranno gli elementi di indice 4, 5, 6, 7, 8, 9
 - "ron de"

Porzioni (*slice*) di una stringa (2)

- Entrambi gli indici usati con l'operatore slice sono opzionali
 - Se il primo indice viene omesso, è sottinteso il primo carattere della stringa (indice 0)
 - Se il secondo indice viene omesso, saranno inclusi tutti gli elementi fino all'ultimo

- Esempi
 - nome[: 6]
 - Inclusi i caratteri dal primo al 6 (escluso)
 - nome[6 :]
 - Inclusi gli elementi dal 6 (compreso) fino alla fine della stringa
 - nome[:]
 - Tutti gli elementi, dal primo all'ultimo (ne fa una copia, operazione inutile per le stringhe)

Porzioni con «passo» diverso da 1

- Nelle porzioni è possibile indicare un terzo argomento
 - `stringa[start : stop : step]`
 - Es.: `nome[2 : 8 : 2]` restituisce una stringa contenente i caratteri di indice 2, 4, 6 → «prn»
 - `nome[::2]` tutti i caratteri di indice pari
 - `nome[1 ::2]` tutti i caratteri di indice dispari
- E se lo **step** è negativo?
 - Intuitivamente, invece di spostarmi in avanti ad ogni passo, mi sposto indietro (e quindi prendo gli elementi in ordine inverso)
 - In questo caso occorre che l'indice **start** sia maggiore di **stop**
 - `nome[9 : 3 : -1]` restituisce «ed nor»
 - `nome[:: -1]` tutta la stringa, in ordine inverso «inorepaP 'ed norepaP»

Operazioni su stringhe



Tabella 7

Operazioni con stringhe

R&S

Enunciato	Risultato	Commento
<pre>string = "Py" string = string + "thon"</pre>	<code>string</code> assume il valore "Python"	L'operatore + applicato a stringhe esegue una concatenazione.
<pre>print("Please" + " enter your name: ")</pre>	Visualizza <code>Please enter your name:</code>	La concatenazione può anche essere usata per comporre stringhe che non trovino posto su una sola riga.
<pre>team = str(49) + "ers"</pre>	<code>team</code> assume il valore "49ers"	Dato che 49 è un numero intero, deve essere convertito in stringa.
<pre>greeting = "H & S" n = len(greeting)</pre>	<code>n</code> assume il valore 5	Ciascuno spazio viene contato come un carattere.
<pre>string = "Sally" ch = string[1]</pre>	<code>ch</code> assume il valore "a"	Si noti che la posizione iniziale è 0.
<pre>last = string[len(string) - 1]</pre>	<code>a last</code> viene assegnata una stringa che contiene l'ultimo carattere di <code>string</code>	L'ultimo carattere si trova nella posizione <code>len(string) - 1</code> .

Caratteri

- I caratteri sono memorizzati come valori interi
 - Vedere il subset ASCII nella tabella Unicode, Appendice A
 - Per esempio, la lettera 'H' ha il valore ASCII 72
- Python usa i caratteri **Unicode**
 - Unicode definisce oltre 100,000 caratteri
 - Unicode è stato creato capace di codificare il testo sostanzialmente in tutte le lingue scritte
 - <https://home.unicode.org/> e <http://www.unicode.org/charts/>

Codici ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Source: www.LookupTables.com

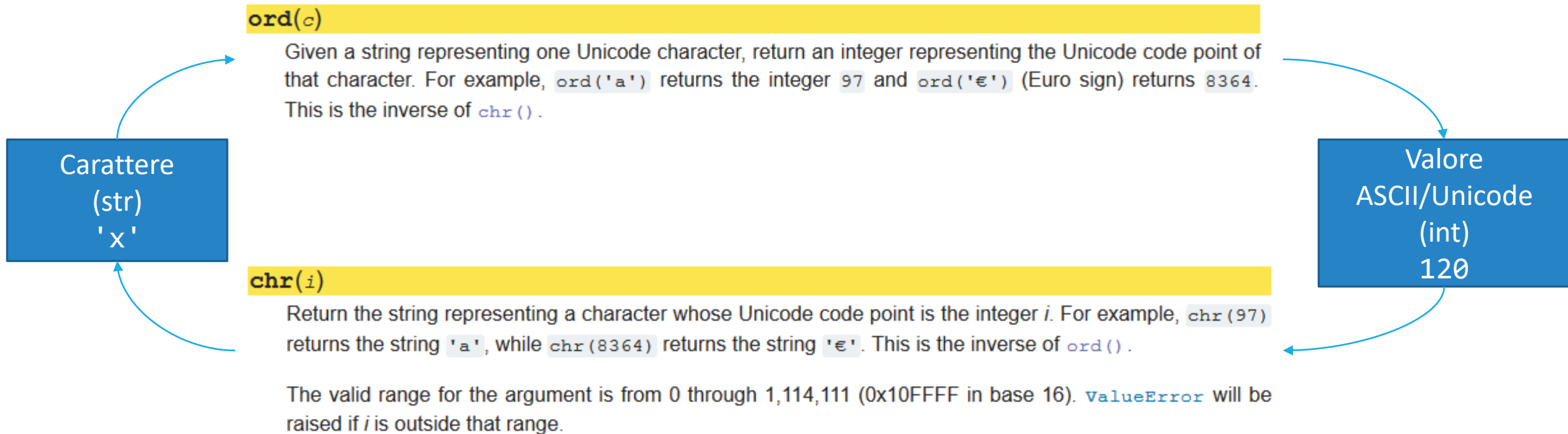
For Unicode characters see: <https://unicode-table.com/>

Codici Unicode

```
0061    'a'; LATIN SMALL LETTER A
0062    'b'; LATIN SMALL LETTER B
0063    'c'; LATIN SMALL LETTER C
...
007B    '{'; LEFT CURLY BRACKET
...
2167    'VIII'; ROMAN NUMERAL EIGHT
2168    'IX'; ROMAN NUMERAL NINE
...
265E    '♞'; BLACK CHESS KNIGHT
265F    '♟'; BLACK CHESS PAWN
...
1F600   '😄'; GRINNING FACE
1F609   '😏'; WINKING FACE
...
```

For Unicode characters see: <https://unicode-table.com/>

Conversione dei caratteri



Funzioni vs. metodi

- Python è un linguaggio orientato agli oggetti (object-oriented) e tutti i valori sono *oggetti*.
 - La programmazione object-oriented è fuori dagli obiettivi del corso, ne vedremo solamente alcuni aspetti pratici
- Ogni oggetto può avere *metodi*, ossia funzioni che possono essere chiamate su *quegli oggetti specifici*, usando la sintassi `object.method()`
- Esempio: tutte le stringhe hanno il metodo `upper()` che restituisce una nuova stringa con i caratteri maiuscoli

```
name = "John Smith"
# Sets uppercaseName to "JOHN SMITH"
uppercaseName = name.upper()
```

Funzioni vs. metodi – cosa ricordare

FUNZIONI

- Le funzioni sono **generali** e possono accettare argomenti di diverso tipo
- Le funzioni sono chiamate **direttamente, con un elenco di parametri**
 - `func(param)`
- Le funzioni restituiscono un risultato che può essere salvato in una variabile
 - `result = func(param)`

METODI

- Diversi metodi sono **specifici** per diversi tipi di oggetti
 - Tutte le stringhe hanno un gruppo di metodi
 - Tutti gli interi hanno un gruppo di metodi
 - ...
- I metodi sono chiamati con la **notazione del punto (dot-syntax)**
 - `object.method()`
- I metodi restituiscono un risultato che può essere salvato in una variabile
 - `result = obj.method()`

Alcuni utili metodi delle stringhe

Tabella 8
Metodi utili per elaborare
stringhe

Metodo	Restituisce
<code>s.lower()</code>	La versione minuscola della stringa <i>s</i> .
<code>s.upper()</code>	La versione maiuscola della stringa <i>s</i> .
<code>s.replace(<i>old</i>, <i>new</i>)</code>	Una nuova versione della stringa <i>s</i> nella quale ogni occorrenza della sottostringa <i>old</i> è stata sostituita dalla stringa <i>new</i> .

Una lista completa di tutti i metodi per lavorare sulle stringhe è:

- <https://www.programiz.com/python-programming/methods/string>
- <https://docs.python.org/3/library/stdtypes.html#string-methods>

Sequenze di *escape*

- Come visualizzare le virgolette?

- Anteporre a " il carattere "\", all'interno della stringa virgolettata
`print("He said \"Hello\"")`

- Come visualizzare il backslash?

- Anteporre a \ un altro \
`print("C:\\Temp\\Secret.txt")`

- Caratteri speciali nelle stringhe

- Andare a capo con '\n'
`print("*\n**\n***")`

```
*  
**  
***
```

Sequenze di escape

Escape Sequence	Description
<code>\newline</code>	Backslash and newline ignored
<code>\\</code>	Backslash
<code>\'</code>	Single quote
<code>\"</code>	Double quote
<code>\a</code>	ASCII Bell
<code>\b</code>	ASCII Backspace
<code>\f</code>	ASCII Formfeed
<code>\n</code>	ASCII Linefeed
<code>\r</code>	ASCII Carriage Return
<code>\t</code>	ASCII Horizontal Tab
<code>\v</code>	ASCII Vertical Tab
<code>\ooo</code>	Character with octal value ooo
<code>\xHH</code>	Character with hexadecimal value HH

<https://www.programiz.com/python-programming/string>

Sequenze di per caratteri Unicode

- Tutti i caratteri Unicode si possono rappresentare
 - Includendoli direttamente nella stringa
 - '♠'
 - Attraverso i codici di escape `\uxxxx` (dove `xxxx` è un codice esadecimale su 16 bit)
 - '\u265e'
 - '\u2620'
 - '\u0420\u043e\u0441\u0441\u0438\u044f'
 - '\u0395\u03bb\u03bb\u03ac\u03c2'
 - Attraverso i codici di escape `\Uxxxxxxxx` (dove `xxxxxxxx` è un codice esadecimale su 32 bit)
 - '\U0001f600'

Sommario

Sommario: variabili

- Un 'valore' è una zona di memoria contenente dei dati di un certo tipo
- Una 'variabile' è nome usato in un programma per riferirsi ad un certo valore.
- Definendo una variabile, bisogna specificare il valore iniziale.
- Per convenzione, i nomi delle variabili iniziano con la minuscola.
- Un'istruzione di assegnazione fa sì che la variabile faccia riferimento ad un nuovo valore (solitamente dello stesso tipo), dimenticando quello precedente.

Sommario: operatori

- L'operatore di assegnazione = non indica l'uguaglianza matematica.
- Le variabili i cui valori iniziali non vanno modificati ('costanti') sono, per convenzione, nominate con caratteri completamente maiuscoli.
- L'operatore / opera una divisione che porta ad un risultato che può avere valore frazionario.
- L'operatore // opera una divisione intera, il resto viene ignorato.
- L'operatore % calcola il resto di una divisione.

Sommario: Python

- Le librerie Python dichiarano varie funzioni matematiche, come `sqrt()` e `abs()`
- Si può convertire tra interi, numeri a virgola mobile e stringhe usando le rispettive funzioni: `int()`, `float()`, `str()`
- La conversione può generare errori, qualora la stringa da convertire non sia nel formato corretto
- Le librerie Python sono raggruppate in moduli. Si usa l'enunciato `import` per utilizzare i metodi da un modulo.

Sommario: stringhe

- Le stringhe sono sequenze di caratteri
- La funzione `len()` restituisce il numero di caratteri in una stringa
- Si usa l'operatore `+` per concatenare le stringhe; ossia per metterle assieme ottenendo una stringa più lunga
- Per la concatenazione, l'operatore `+` impone che entrambi gli argomenti siano stringhe. I numeri vanno convertiti in stringhe con la funzione `str()`

Sommario: stringhe

- Si usa l'operatore `[pos]` per estrarre gli elementi (singoli caratteri) dalle stringhe, oppure `[a:b:c]` per estrarre una porzione (slice) di caratteri
- Gli indici dei caratteri delle stringhe sono calcolati partendo da 0
- La libreria standard contiene numerose funzioni di manipolazione delle stringhe
- Tutti i metodi e le funzioni sulle stringhe restituiscono sempre una nuova stringa, non modificano mai quella precedente
- Per convertire i singoli caratteri dal formato stringa al valore numerico ASCII/Unicode utilizzare le funzioni `ord()` e `chr()`