Università degli Studi di Cagliari
Corsi di Laurea in Ingegneria Chimica e Ingegneria Meccanica

FONDAMENTI DI INFORMATICA

http://people.unica.it/gianlucamarcialis

A.A. 2019/2020

Docente: **Gian Luca Marcialis**

LINGUAGGIO PYTHON
Tipi di dati

Sommario

- Classificazione
- Stringhe
  - Caratteri
- Numeri
- Liste
- Dizionari

Variabili e tipi di dati

- Quando in Python si procede con un'istruzione di assegnazione, nel contempo si effettua una sorta di «dichiarazione di tipo»

- Esempio: `a=5`
  - Stiamo memorizzando un valore appartenente al tipo di dato «intero» all'interno della variabile 'a'

- Esistono altri tipi di dato in Python

Premessa

- Come abbiamo visto, il calcolatore rappresenta qualsiasi informazione come una sequenza di bit (stringa binaria)
- Per potere manipolare l'informazione ad un più alto livello, conferendole una semantica, occorre strutturare questa sequenza in modo tale da farle rappresentare:
  - Valori numerici con e senza segno
  - Sequenze di caratteri che formano parole o frasi (stringhe)
  - Combinazioni ordinate dei tipi di valori di cui sopra (tuple, liste, dizionari)

Perché i «tipi di dato»

- Pensiamo alle immagini viste nel cap. 1
  - Ci serve spazio per l'header (intestazione)
  - Ci serve spazio per una sequenza di byte che rappresentano i livelli di grigio
- Il calcolatore raggruppa sequenze di byte in funzione dello spazio fisico disponibile
  - Pensate proprio a sequenze di bit incastonati in caselle fisiche
  - Se lo spazio fisico è finito, potremmo non essere in grado di immagazzinare certe informazioni
- Il risultato è che l'organizzazione di sequenze di byte permette di manipolare dati di dimensione molto maggiore di quella dei singoli byte!
- Da qui la necessità di conoscere prima a che *tipo di dato* ci stiamo riferendo per consentire al calcolatore di verificare ed organizzare lo spazio fisico a sua disposizione

Classificazione dei tipi di dati

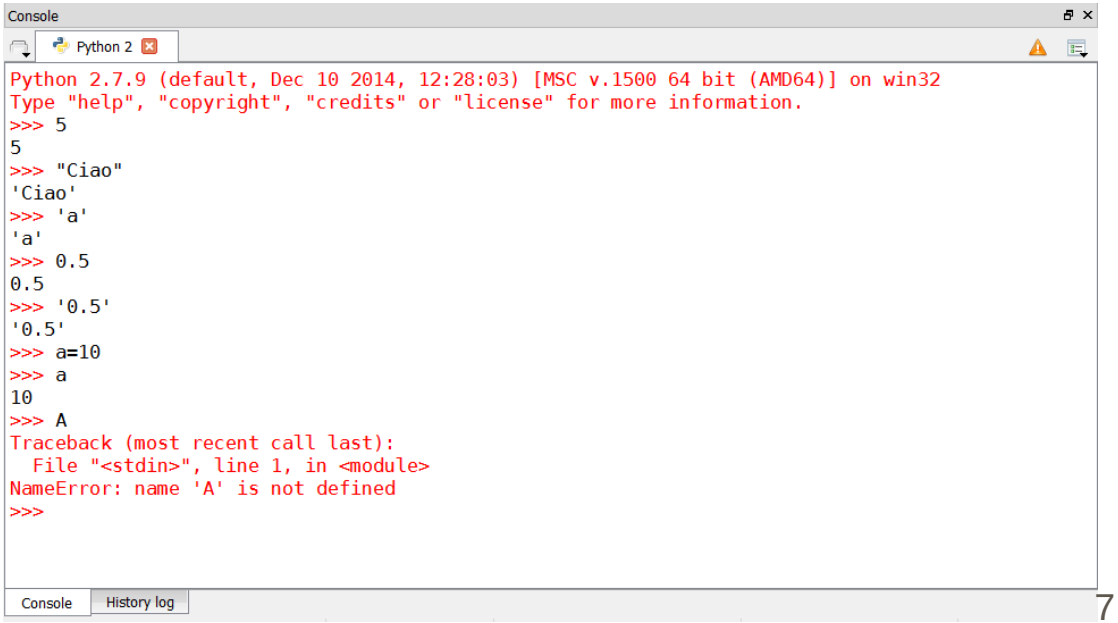
- Tipi semplici
  - Valori numerici
    - Es. per rappresentare una misura di velocità, una temperatura, il valore del livello di grigio o di colore in un'immagine
- Tipi strutturati
  - Indicano "raggruppamenti" di tipi semplici (e, a loro volta, di tipi strutturati)
- In Python esistono tipi semplici e strutturati predefiniti

Primi tipi predefiniti

Tipi di dati	Nome del tipo di dato in Python	Esempi
Booleani	`bool(int)`	False, True (0, not 0)
Interi	`int`	-1, 0, 1, 2
Numeri reali	`float`	-0.55, .3333, 3.14, 6.0, 3.14e0, -2.5e3
Stringhe di caratteri	`str`	‘Ciao’, ‘’, “66”, “A”, “0Ae;”

La console (o «shell») dell’interprete Python restituisce sempre la stessa istanza del tipo di dato inserito.

Esercizio: provare sull’ambiente Python.



L'operatore type() per la verifica del tipo

Restituisce il tipo di una certa istanza (costante o variabile)
Sintassi: `type(<istanza>)`

Tipo	Esempio di istanza	Verifica di tipo (provate)
bool → booleano	True, False	`type(True)==bool`
int → intero	2, -1, 0	`type(-1)==int`
float → reale	2.1, -3.54543	`type(2.1)==float`
str → stringa	'ciao', 'buon2.1.''	`type('ciao')==str`

Istanza costante

```
>>>type(2.5)
float
>>>type('ale')==str
True
```

Istanza variabile

```
>>>v=2.5
>>>type(v)
float
>>>type(v)==str
False
```

Conversioni di tipo

Funzione di conversione	Esempio	Risultato
`int`(<numero o stringa>)	`int(3.88)`	`3`
	`int('33')`	`33`
`float`(<numero o stringa>)	`float(22)`	`22.0`
	`float('22.6')`	`22.6`
`str`(<tipo>)	`str(99)`	`'99'`
`bool`(<tipo>)	`bool(0)`	`False`

Troncamento

Funzione	Esempio	Risultato
`round`(<numero>,cifre)	`round(3.56)`	`4.0`
	`round(3.55,1)`	`3.6`

Esercizio

- Scrivere un programma Python che, letti da tastiera due numeri interi come dividendo e divisore, stampi a video il RESTO della divisione senza usare l'operatore %

Conversioni di tipo

```
>>>float(2)
2.0
>>>int(2.6)
2
>>>bool(1)
True
>>>str(2.6)
‘2.6’
```

```
>>>float(‘3.4’)
3.4
>>>int(‘3.4’)
3
>>>bool(‘ciao’)
True
>>>str(False)
‘False’
```

Troncamento

- La conversione da float a int consente di prelevare la parte intera di un numero reale
- Tuttavia potrebbe servirci un'approssimazione con un numero inferiore di cifre nella parte frazionaria

Funzione	Esempio	Risultato
`round`(<numero>,cifre)	`round(3.56)`	`4.0`
	`round(3.56,1)`	`3.6`

- Notate che l'operatore non effettua alcuna conversione di tipo, ma semplicemente un troncamento ad un numero prefissato di cifre frazionarie

Manipolazione di valori numerici

- Int e float condividono le stesse operazioni fondamentali ma l'uscita è differente
  - **Esercizio**. Scrivere un programma Python che stampi a video il risultato della divisione tra due valori interi e gli stessi valori rappresentati come float
- Su tutti i tipi numerici sono definiti operatori di confronto:
  - `==` uguaglianza
    - (es. `a==b` restituisce il valore `False` se `a` e `b` sono diversi)
  - `!=` diversità (es. `a!=b`)
  - `<`, `>` minore, maggiore (es. `a>b`)
  - `<=`, `>=` minore o uguale, maggiore o uguale (es. `a<=b`)

Espressioni aritmetiche

Operatore	Descrizione	Sintassi
-	Cambio di segno	`-numero`
**	Elevamento a potenza	`base ** esponente`
*	Moltiplicazione	`moltiplicando * moltiplicatore`
/	Divisione	`dividendo / divisore`
//	Quoziente	`dividendo // divisore`
%	Resto o modulo	`dividendo % divisore`
+	Addizione	`addendo1 + addendo2`
-	Sottrazione	`sottraendo - sottrattore`

Precedenze:

- Potenza; moltiplicazione, divisione; addizione e sottrazione
- Le operazioni di uguale precedenza sono associative a sinistra, con l'eccezione dell'elevamento a potenza associativa a destra
- Per modificare le precedenze si usano le parentesi

La «libreria» delle funzioni matematiche

- Una «libreria» è un file che contiene un insieme di funzioni finalizzate, per esempio, a potenziare un certo tipo
- Quelle matematiche si importano con il modulo `math`

```
import math
dir(math)        #elenco delle funzioni disponibili
```

- Esempi:

```
math.pi          #valore pi-greca
math.sqrt(2)     #radice quadrata
math.cos(a)      #coseno di a in radianti
```

- Per importare solo una selezione di funzioni:

```
from math import pi, sqrt, cos
```

Altre funzioni matematiche

Funzione	Descrizione	Sintassi
`cos(angolo)`	Coseno	`math.cos(angolo)`
`sin(angolo)`	Seno	`math.sin(angolo)`
`exp(x)`	Espondenziale e^x	`math.exp(x)`
`log(x)`	Logaritmo base `e`	`math.log(x)`
`log10(x)`	Logaritmo base 10	`math.log10(x)`
`fabs(x)`	Valore assoluto	`math.fabs(x)`

Se utilizziamo il costrutto:

```
from math import sin, exp      #per esempio
```

invocare queste funzioni non richiede il suffisso `math.`

Esercizio

- Scrivere un programma Python che, leggendo da tastiera il valore del raggio di un cerchio, stampi a video il valore della circonferenza e dell'area.
- Suggerimento: usare ciò che serve importandolo dalla libreria `math`

Soluzione

```
"""
Programma per la stampa a video di circonferenza ed area di un cerchio
Input: raggio del cerchio
Output: stampa a video circonferenza ed area

@author: Gian Luca
"""

from math import pi #importiamo dalla libreria solo ciò che ci serve

raggio=input("Inserire il raggio del cerchio.")


circonferenza=2 * pi * raggio
area=pi * raggio ** 2


print("La circonferenza vale " + str(circonferenza))
print("e l\' area vale " + str(area))
```

Stringhe di caratteri

- Una stringa è una sequenza di caratteri:
  - Lettere
  - Numeri
  - Segni di interpunzione
  - Caratteri speciali anche chiamati **sequenze di escape** (‘a capo’, ‘tabulazione’, ‘fine file’)
- In Python **deve essere** racchiusa da una coppia di virgolette doppie o singole
  - `‘‘Ciao’’, ‘Ciao’`
  - `‘12; come va? \n\t’`
  - `‘’` ➔ `carattere` **`vuoto`**
  - `Ciao` **non è** una stringa (mancano le virgolette)
- Per stampare una stringa si usa la funzione `print`

Numeri e set di caratteri

- Come sappiamo nel calcolatore ogni carattere alfanumerico ha un suo codice ASCII
- In Python risaliamo ad esso tramite la funzione `ord()`, mentre il carattere corrispondente ad un certo codice viene dato tramite `chr()`
- Esempi:

```
ord('a') → 97
ord('A') → 65
chr(65) → 'A'
chr(97) → 'a'
```

Lettura di stringhe da tastiera

- Funzione **`raw_input`**:
- Uso:

```
<variabile> = raw_input(<stringa-di-commento>)
```

- Esempio:

```
stringa = raw_input('Inserisci una stringa:')
```

Attenzione Python 2.7 vs. Python 3.6

- Le funzioni raw_input() di Python 2.7 e input() di Python 3.6 si equivalgono
- Python 2.7
  - `Stringa=raw_input('Inserisci una stringa:')`
- Python 3.6
  - `Stringa=input('Inserisci una stringa:')`
- D'ora in avanti useremo sempre `raw_input()` per la lettura di dati da tastiera per la 2.7; coloro che hanno la versione 3.6 potranno usare con il medesimo risultato `input()`

Esercizio

- Scrivere un programma Python che legga a video un carattere alfabetico minuscolo e stampi il corrispondente maiuscolo

- Suggerimento. Si osservi la tabella ASCII (standard) a lato

Dec	Sym	Dec	Char	Dec	Char	Dec	Char
0	NUL	32		64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	TAB	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	□

Soluzione

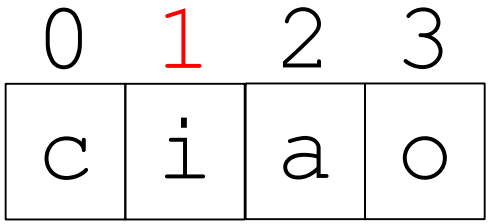
```
"""
Programma per la conversione minuscolo-maiuscolo
Stampa a video il corrispondente maiuscolo di un carattere dato
Input: un carattere c (immesso tra virgolette)
Output: stampa a video del corrispondente maiuscolo
"""

c=raw_input("Digita un carattere minuscolo:\n") #immissione di un carattere

codice_c=ord(c) #calcola il codice ASCII del carattere di input
codice_C=codice_c-32 #calcola il codice ASCII del corrispondente maiuscolo

C=chr(codice_C)

print "Il corrispondente maiuscolo di " + c + " è " + C + ".\n"
```

Sequenze di escape (caratteri speciali)

Sequenza di escape	Significato
\b	Backspace
\n	A capo (newline)
\t	Tabulazione orizzontale
\\	Carattere \
\’	Virgoletta semplice
\”	Virgolette doppie

Indicizzazione delle stringhe

- I caratteri di ciascuna stringa sono indicizzati. Ad esempio questa stringa:




```
>>>s='ciao'
>>>s[1]
'i'
>>>s[1:3] #indicizzazione per intervallo
'ia'
```

Immutabilità dei caratteri di una stringa

- Una volta assegnata ad una variabile, non è possibile riassegnare il singolo carattere

```
>>> stringa='ciao' #assegno una stringa ad una variabile
>>> stringa[0]='m' #voglio cambiare la 'c' con la 'm'
```

ERRORE!!!

- Per generare la stringa 'miao' da 'ciao' devo fare così:

```
>>>stringa='m'+stringa[1:]
```

Operazioni con le stringhe

- Concatenazione

```
'Ciao  ' + ' sono ' + ' Gian Luca!\n '

' ' * 10 + ''Ciao!!!\n'
```

- Concatenazione iterativa

```
numspazi=input('Dimmi quanti spazi:\n')
' ' * numspazi + 'Ciao!!!'
```

Altre operazioni con le stringhe

Data una stringa `s`	Azione
`len(s)`	Il numero di caratteri della stringa `s`
`s.isalpha()`	*True* se `s` è fatta di caratteri alfabetici
`s.isdigit()`	*True* se `s` è fatta di caratteri numerici
`s.lower(), s.upper()`	Stringa fatta solo di caratteri minuscoli o maiuscoli
`s.count(sub)`	Numero di occorrenze della sottostringa `sub` in `s`
`c [not] in s`	*True* se `c` [non] è una sottostringa `di s`

```
>>>'ciao'.isalpha()      >>>'ciao'.upper()
>>>'2.5'.isdigit()       >>>'c' in 'ciao'
>>>'buonanotte'.count('n')
```

Esercizio

- Scrivere un programma Python che, letta una stringa da tastiera, stampi a video, nell'ordine:
  - La lunghezza della stringa
  - La stringa con tutti i caratteri maiuscoli
  - Generi una stringa in cui il primo e l'ultimo carattere sono permutati
- Esempio. Se si legge da tastiera la stringa 'buonanotte', il programma stamperà a video:
  - 10
  - BUONANOTTE
  - euonanottb

Soluzione

```
#legge una stringa da tastiera
stringa=raw_input('Immetti una stringa: ')
#stampa lunghezza e stringa in maiuscolo
L=len(stringa)
print L
print stringa.upper()
#assegna nuova stringa secondo specifiche
stringa_nuova=stringa[L-1] + stringa[1:L-1]
+ stringa[0]
#stampa la stringa generata
print stringa_nuova
```

Le liste

- Le liste sono una sequenza ordinata di elementi, anche eterogenei, separati da virgole e racchiusi tra due parentesi quadre [ ]
- Esempi di liste

```
listaDiStringhe= [‘Questa’, ‘è’, ‘una’, ‘lista’, ‘di’, ‘stringhe.’]

listaDiNumeri=[12, 30.59, -5, .4, -300.2]

listaMista=[12, ‘Buongiorno’, 54, 6.2, ‘a tutti’]

listaVuota=[]

listaDiListe=[listaDiStringhe, listaDiNumeri, listaMista, listaVuota]
```

Elementi di una lista

- L'elemento generico si identifica attraverso la sintassi

```
nomeLista[indicePosizione]
```

- Esempio.

```
l=[23, 5, 12]
print l[0] #stampa il primo elemento
print l[1] #stampa il secondo elemento
print l[2] #stampa il terzo elemento
print l[3] #errore!
```

Assegnazione di una lista ad una variabile

```
>> l=[3, 5, 6]
>> print l
[3, 5, 6]
>> l=['ciao', 'come', 'stai?']
>> print l
['ciao', 'come', 'stai?']
>> print l[1]
come
>> l[1]=12
>> print l #quale sarà l'effetto?
```

Liste e stringhe

- Provate la seguente:

```
>>>stringa='ciao'
>>>lista=list(stringa)
```

- Ora stampate la variabile `lista`

- Cosa è successo?

Liste e stringhe

- Avete stampato:

```
['c', 'i', 'a', 'o']
```

- L'operatore `list()` ha convertito la stringa in una lista di stringhe formate dai singoli caratteri

- A questo punto proviamo con:

```
s=str(l)
print s
```

Liste e stringhe

- Abbiamo ottenuto:

`'['c', 'i', 'a', 'o']'`

- L'operatore `str()` ha trasformato la nostra lista in una stringa, incluse le parentesi quadre!

- C'è dunque un legame tra stringhe e liste… come si stamperanno i singoli caratteri di `s`?
  - Esattamente come i singoli elementi di una lista!

Esercizio

- Assegnare la stringa `'ciao'` alla variabile `saluto`
- Stampare a video la variabile `saluto`
- Stampare a video ogni singolo carattere, separato da spazio, contenuto nella variabile `saluto`

Soluzione:

```
>>>saluto='ciao'
>>>print saluto
>>>print saluto[0], saluto[1], saluto[2], saluto[3]
>>>print   saluto[0]+'   '+saluto[1]+'   '+saluto[2]+'
'+saluto[3]
```

Domanda

- Qual è la differenza tra

```
>>>print saluto[0], saluto[1], saluto[2], saluto[3]
```

e

```
>>>print    saluto[0]+’    ‘+saluto[1]+’    ‘+saluto[2]+’    ‘+saluto[3]
```

- Per scoprirlo, assegnare ad una variabile l’argomento della prima `print`, e ad una seconda variabile l’argometno della seconda `print`:

```
a= saluto[0], saluto[1], saluto[2], saluto[3]
b= saluto[0]+’ ‘+saluto[1]+’ ‘+saluto[2]+’ ‘+saluto[3]
```

Tuple

```
>>>a
('c', 'i', 'a', 'o')    #è una tupla di
                        #caratteri
>>>b
'c i a o' #è una stringa
```

Le tuple sono immutabili, come le stringhe.

Presentano tutte le caratteristiche delle stringhe, sono indicizzate allo stesso modo

```
>>>a[0]==b[0]                >>>len(a)==len(b)
True                        False #Perché?
```

Liste e stringhe

- Gli elementi di una stringa e di una lista presentano affinità, ma la lista è più duttile
- Data una lista `l`, per esempio, possiamo usare `len(l)` per conoscere il numero di elementi
- L'indicizzazione segue le stesse leggi
- MA i singoli elementi di una lista possono essere riassegnati, mentre quelli delle stringhe, come delle tuple, no

Estrazione di sottostringhe/sottoliste

- E' sufficiente inserire l'intervallo di indici di interesse nel formato `i:j[:h]` dove
  - `i` ➔ indice iniziale; `j` ➔ indice finale; `h` ➔ passo (1 se omesso)

```
>>>s='buonanotte'
>>>s[2:5] #l'indice finale è escluso
'ona'
>>>l=['ora', 1, -3, 'tarda']
>>>l[0:3:2]
['ora', -3] #'ora' è l[0], -3 è l[2]
```

Scorrere una lista (stringa) al contrario

- Si usano i valori negativi:

```
>>>l=[0,1,2,3,4,5,6,7,8,9]
>>>l[-1]
9
>>>l[-2]
8
>>>l[-1:-10:-1] #valuta da l[-1] a l[-9]
[9,8,7,6,5,4,3,2,1]
```

Il metodo «join» di stringa

- Data una stringa `s`, ed una lista `l` di stringhe, scrivendo: `s.join(l)`

concateniamo le stringhe di `l` usando `s` come elemento separatore tra esse.

- Esempio:

```
>>>s=‘ ’ #carattere spazio
>>>l=[‘ciao’,‘come’, ‘stai?’]
>>>print s.join(l)
ciao come stai?
>>>print ‘spazio’.join(l) #provate!!!
>>>print ‘ ’.join(l) #provate!!!
>>>print ‘’.join(l) #provate!!!
```

Esercizio

- Data una lista di caratteri `l`, generare una stringa ottenuta dalla concatenazione dei singoli caratteri.
- Esempio. Data:

`l=['c', 'i', 'a', 'o']`

Ottenere la stringa `'ciao'`.

Soluzione?

Operazioni su liste

Date le liste `l`, `l1`, `l2` e una stringa `s`	Effetto
`l=[]`	Crea una lista vuota e l'assegna ad `l`
`l=l1+l2`	Assegna ad `l` la concatenazione di `l1` ed `l2`
`n=len(l)`	Assegna ad `n` il numero di elementi di `l`
`l[index]`	L'elemento in posizione `index` della lista
`l=list(stringa)`	Assegna ad `l` una lista composta dai singoli caratteri di stringa
`l=[v] * n`	Assegna ad `l` una lista replicando `n` volte l'elemento `v`
`x [not] in l`	*True* se `x` [non] è presente in `l`
`l=s.split(sep)`	Assegna ad `l` una lista di stringhe che usano `sep` come stringa di separazione
`s=sep.join(l)`	Assegna ad `s` una stringa ottenuta concatenando le stringhe di `l` ed usando `sep` come stringa separatrice

Esercizi su liste e stringhe

- Ora possiamo capire meglio qualche operatore di stringa e valutarne l'utilità

```
>>>s='ciao come stai?'
>>>l=s.split()
>>>print l
['ciao', 'come', 'stai?']
>>>s='3/10/1934'
>>>print s.split('/')
```

Operazioni su liste

- Le liste supportano l'operatore di concatenazione :

```
>>>l=[]                  #[]
>>>l=l+[3]               #[3]
>>>l=l+['coda']          #[3, 'coda']
>>>l=['testa']+l         #['testa', 3, 'coda']
>>>l2=[l, len(l)]
>>>l2
```

cosa accadrà?

Esercizio

- Scrivere una sequenza di istruzioni (un algoritmo) Python che, data una stringa `s`, permuti l'ultimo ed il primo carattere.
- Esempio: data `'ciao'`, ottenere `'oiac'`.
- Abbiamo già svolto questo esercizio, ma ora si usi quel che si è imparato sulle liste.

- Soluzione?

Esercizio

- Si consideri la seguente stringa:

`'Paolo nato nel 2001 a Cagliari.'`

- Scrivere una sequenza di istruzioni Python che stampi a video la stringa:

`'Paolo ha 15 anni.'`

sulla base del nome e dell'anno estratti dalla stringa data.

Esercizio

- Generare una lista di dieci valori interi pari a 0

- Leggere un valore intero da tastiera pari a uno degli indici della lista di cui sopra (quindi da 0 a 9) e aggiornare il valore nella relativa posizione sommando ad esso il valore 1.
  - Usare `raw_input()` per la 2.7 o `input()` per la 3.6

Dizionari in Python

- Simili alle liste, associano ad ogni valore (**value**) un attributo chiamato chiave (**key**):

$$d = \{k_1 : v_1, k_2 : v_2, \dots, k_n : v_n\}$$

- Esempio:

```
>>>contatti={'Marcialis': 5893, 'Fumera': 5754, 'Giacinto': 5752}
```

- La variabile contatti presenta una chiave separata dal suo valore tramite ':'.
- Per accedere ad un dato valore si usa la sua chiave.

```
>>>contatti['Marcialis']
```

Operazioni e metodi sui dizionari

Dato un dizionario `d`	Effetto
`d={}`	Assegna alla variabile `d` il dizionario vuoto `{}`
`len(d)`	Restituisce il numero di coppie chiave:valore in `d`
`key [not] in d`	*True* se `key` [non] è chiave di `d`
`d[key]=value`	Assegna `value` alla chiave `key` se presente o ne aggiunge una nuova
`v=d[key]`	Assegna a `v` il valore associato a `key`
`d.pop(key)`	Elimina la chiave `key` e relativo valore dal dizionario
`d.values()`	Restituisce la lista dei valori del dizionario
`d.keys()`	Restituisce la lista delle chiavi del dizionario

Esercizio

- Generare un **dizionario** con tre chiavi: ‘Nome’, ‘Cognome’ e ‘Numero di telefono’. Ogni chiave è associata ad una relativa **lista** di valori (due stringhe ed un intero). Per esempio:

Nome	Cognome	Numero di telefono
Gian Luca	Marcialis	5893
Fabio	Roli	5779
Giorgio	Giacinto	5752

- Stampare a video la seconda riga della tabella in modo da visualizzare:

```
Fabio     Roli      5779
```

- Cambiare il numero di telefono di `Giacinto` in `5754`.

Soluzione

```
#Domanda 1
d={"Nome":["Gian Luca", "Giorgio", "Fabio"], "Cognome": ["Marcialis", "Giacinto", "Roli"], "Numero di telefono": [5893, 5752, 5779]}

#Domanda 2
s=""
l=d["Nome"]
s=s+l[2]
```

…completate voi

Per saperne di più

- K.A. Lambert, *Programmazione in Python*, Capp. 2, 4, 5, Apogeo