Fondamenti di programmazione

Canale A-L - Prof. Sterbini

AA 25-26

Lezione 2

RECAP LEZ1

- organizzazione del corso
- un po' di storia
- Numeri interi (int),
- Numeri decimali (float)

In [1…
```
(-1+6)/5**2*10+(10*10)**0.5+1+3
```

Out[1…
```
16.0
```

FDPLEZ2

Momento Wooclap.com

A che punto siete con l'installazione di Anaconda e IDE?

Quanto fa questa espressione?

Numeri con la virgola (di tipo `float`)

Sono codificati con un numero di bit fisso (1 o 2 word)
(Vedi standard IEEE 754)

Esempio: word da 64 bit (per CPU a 64 bit)

segno (1 bit)	esponente (11 bit)	mantissa (52 bit)
0=+ 1=-	esponente	cifre significative

Hanno un numero fisso di bit nella mantissa, quindi la precisione
(cioè il numero di cifre corrette) è limitata

- una word -> 64 bit -> 52 bit di mantissa -> circa 16 cifre decimali
- due word -> 128 bit -> 112 bit di mantissa -> circa 33 cifre decimali

Esempio (con 1 bit di segno, 3 bit di esponente e 4 bit di mantissa)

S	E	E	E	M	M	M	M
1	0	1	0	1	0	1	1

- Il numero si trova calcolando $$(-1)^{segno} * 2^{esponente+offset} * 1.mantissa$$
- segno = 1 -> numero negativo
- esponente = 010 -> in complemento a 2 vale 2
- per comodità supponiamo che l'offset sia 0 (dipende in realtà dallo standard)
- mantissa = 1011 -> viene aggiunta un 1 (e una virgola) a sinistra
  - 1.1011 viene convertito in decimale sapendo che i pesi, andando verso destra si dividono per 2 -> (0.1 = 1/2, 0.01 = 1/4, 1/8, 1/16, 1/32)
  - quindi il valore della parte mantissa 1.1011 è $1+1/2+0+1/8+1/16= 27/16 = 1.6875$
- il numero quindi è: ${ }-1 * 2^{2+0} * 1.6875 = -6.75$

In [1…
```
# Vediamo quante cifre decimali possono essere rappresentate da una mantissa di 52 bit
# 2 alla 52 è maggiore di 10 alla 15 e minore di 10 alla 16?
print(' 52 bit sono circa 16 cifre decimali?', 10**15 < 2**52 < 10**16)
print(2**52, 10**15, 10**16)
```

```
 52 bit sono circa 16 cifre decimali? True
4503599627370496 1000000000000000 10000000000000000
```

In [2…
```
# 2 alla 112 è maggiore di 10 alla 13 e minore di 10 alla 34?
print('112 bit sono circa 33 cifre decimali?', 10**33 < 2**112 < 10**34)
```

```
112 bit sono circa 33 cifre decimali? True
```

In [3…
```
# vediamo che succede se stampo 10 alla 15 oppure 10 alla 16
print(10.0**15, 10.0**16) # del primo stampa TUTTE LE CIFRE! il secondo è invece "approssimato"

type(17.5)   # ecco il tipo di un numero con la virgola
```

```
1000000000000000.0 1e+16
```

Out[3…
```
 float
```

In [4…
```
# ESPRESSIONI MATEMATICHE
# posso scrivere espressioni matematiche e calcolarle
```

```
# (valgono le precedenze: PRIMA **, POI * e / E POI + e -)
print('3 + 14 * 5 / 4 = ', 3 + 14 * 5 / 4 )
# prima calcola 14*5=70 poi 70/4=17.5 poi 17.5+3=20.5
```

```
3 + 14 * 5 / 4 =  20.5
```

In [5…
```
print('3 + 5 * 10**2 / 4 = ', 3 + 5 * 10**2 / 4)
# prima calcola 10**2=100 poi 5*100=500 poi 500/4=125.0 poi 125.0+3=128.0
# NOTA la divisione produce un float e da quel momento in poi i risultati sono float
```

```
3 + 5 * 10**2 / 4 =  128.0
```

In [6…
```
# ma se voglio che vengano calcolate in ordine diverso
# basta usare le parentesi tonde
# (SOLO TONDE! le quadre e graffe hanno significato diverso)
print('14 * 5 / (4 + 3) = ', 14 * 5 / (4 + 3))
# prima calcola 14*5=70 poi 4+3=7 poi 70/7=10
# dato che uso la divisione normale ottengo sempre un float
```

```
14 * 5 / (4 + 3) =  10.0
```

Il testo viene chiamato "stringa" (di tipo `str`)

E' formato da sequenze di caratteri

i testi sono **IMMUTABILI** (potete solo crearne di nuovi in memoria)

(ma Python recupera automaticamente la memoria dei vecchi dati non usati)

Ciascun carattere è codificato con la codifica UNICODE che contiene tutti i possibili caratteri (Ebraico, Greco, Arabo, Tamil, Geroglifico, Giapponese ...)

Il testo va racchiuso tra singoli (`'`) o doppi apici (`"`)

In [7…
```
# esempio di un testo
"Paperino andò al mare"
```

Out[7…
```
'Paperino andò al mare'
```

Possiamo estrarre un carattere dalla stringa indicandone la posizione da sinistra (partendo da 0)

Caratteri ->	"	P	a	p	e	r	**i**	n	o	"	
Posizione ->		0	1	2	3	4	**5**	6	7		(indice)

In [8…
```
# estraggo il carattere in posizione 5 (partendo da 0)
A = "Paperino"
A[5]    # leggo il valore che sta all'indice 5
        # tra parentesi quadre metto l'INDICE del carattere che interessa
```

Out[8…
```
'i'
```

In [9…
```
# e ne posso calcolare il codice UNICODE con 'ord' (e il tipo del dato contenuto in A)
ord(A[5]), ord('i'), type(A)
```

Out[9…
```
(105, 105, str)
```

In [1…
```
# oppure posso ottenere il carattere a partire dal suo codice UNICODE con 'chr'
chr(105)
```

Out[1…
```
'i'
```

In [1…
```
# Posso concatenare due o più stringhe ottenendone una nuova con l'operatore '+'
'Paperino' + ' ' + "andò al mare" + ' con Minnie'
```

Out[1…
```
'Paperino andò al mare con Minnie'
```

In [1…
```
# o ripetere più volte la stessa concatenazione con l'operatore '*'
"Minnie " * 5
```

Out[1…
```
'Minnie Minnie Minnie Minnie Minnie '
```

In [1…
```
# Per spezzare la stringa rispetto a un separatore (per default gli spazi)
# si usa il *metodo* 'split' delle stringhe
A = 'Paperino   andò al mare    con Pippo e \n Minnie'
A.split()    # chiedo alla stringa di spezzarsi sugli spazi
# ottengo una lista di frammenti (stringhe)
```

Out[1…
```
['Paperino', 'andò', 'al', 'mare', 'con', 'Pippo', 'e', 'Minnie']
```

In [1…
```
# Alcuni caratteri sono speciali e possono essere indicati con backslash+carattere
accapo    = '\n'      # newline
tab       = '\t'      # tab
backslash = '\\'
doppio_apice = '\"'
apice     = '\''
print('Paperino\nando\'\tal\tmare')    # inserisco accapo, accento/apostrofo e tabulazioni
```

```
Paperino
ando'   al      mare
```

Testo su più righe

Si racchiude in tripli apici `'''` o tripli doppi apici `"""`

Possiamo direttamente usare gli accapi senza inserire esplicitamente `\n`

```
'''Paperino va al mare
con Minnie
e Topolino'''
```

Variabili (nome -> spazio in memoria)

Il programma contiene una/più **"tabella dei nomi"** (namespace) che contiene la corrispondenza tra:

- Nome della variabile
- posizione in memoria del dato

Ad ogni "nome" corrisponde un pezzetto di memoria che contiene la posizione in memoria

Che a sua volta indica DOVE (indirizzo) sta in memoria l'oggetto "contenuto" nella variabile

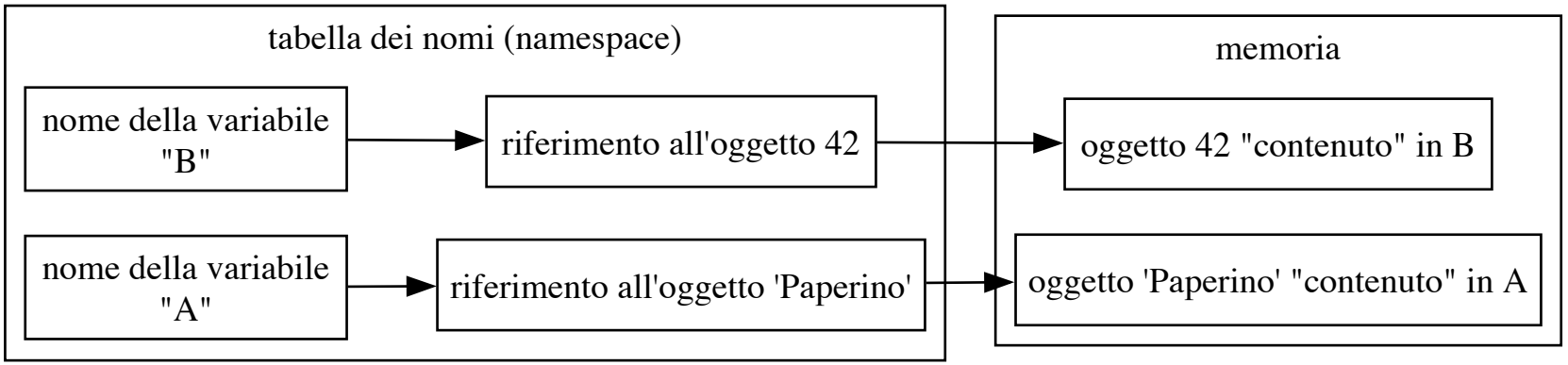
Si chiama "variabile" perchè il suo contenuto può cambiare durante l'esecuzione del programma

In [1…
```
from pygraphviz import AGraph
T = "Quando scriviamo \nA = 'Paperino'  \nB = 42"
A = 'nome della variabile\n"A"'
B = 'nome della variabile\n"B"'
RA = "riferimento all'oggetto 'Paperino'"
RB = "riferimento all'oggetto 42"
OA = 'oggetto \'Paperino\' "contenuto" in A'
OB = 'oggetto 42 "contenuto" in B'
figura = AGraph(directed=True, rankdir='LR')
figura.node_attr.update(shape='box')
figura.add_path([A,RA,OA])
figura.add_path([B,RB,OB])
figura.add_subgraph([A,RA,B,RB],   label='tabella dei nomi (namespace)', cluster=True)
figura.add_subgraph([A,RA,B,RB,OA,OB], label='memoria',   cluster=True)
figura.layout('dot')
```

In [1…
```
print(T)
figura
```

```
Quando scriviamo
A = 'Paperino'
B = 42
```

Out[1…



Perchè questo indirizzamento "indiretto"?

Come si comporta invece un linguaggio compilato come il C?

In **C** la 'tabella dei nomi' esiste **SOLO nel compilatore** che decide una volta per tutte dove mettere le variabili in memoria

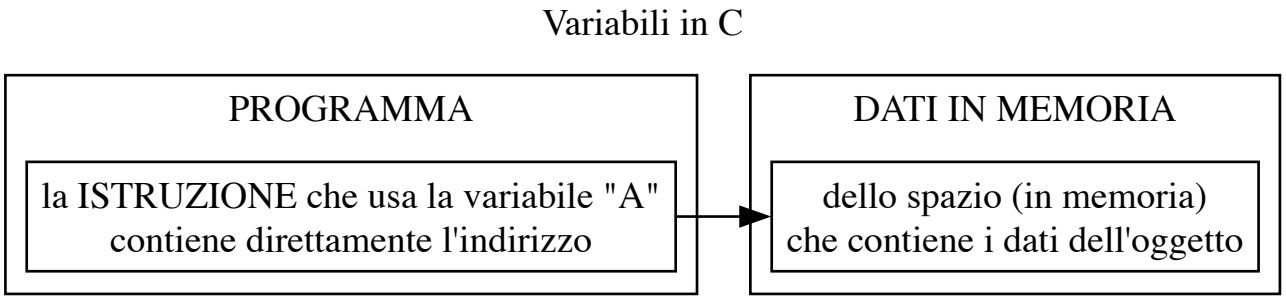
Nel programma eseguibile **i nomi delle variabili non esistono e nel codice oggetto sono direttamente presenti solo le posizioni degli spazi di memoria che contengono i dati**

Questo è molto efficiente (ma rigido)

In [1…
```
from pygraphviz import AGraph
T = "Variabili in C"
A = '''la ISTRUZIONE che usa la variabile "A"
contiene direttamente l'indirizzo'''
O = '''dello spazio (in memoria)
che contiene i dati dell'oggetto'''
figura2 = AGraph(directed=True, rankdir='LR', label=T, labelloc='t')
figura2.node_attr.update(shape='box')
figura2.add_path([A,O])
figura2.add_subgraph([A],cluster=True, label='PROGRAMMA')
figura2.add_subgraph([O],cluster=True, label='DATI IN MEMORIA')
figura2.layout('dot')
```

In [1…
```
figura2
```

Out[1…



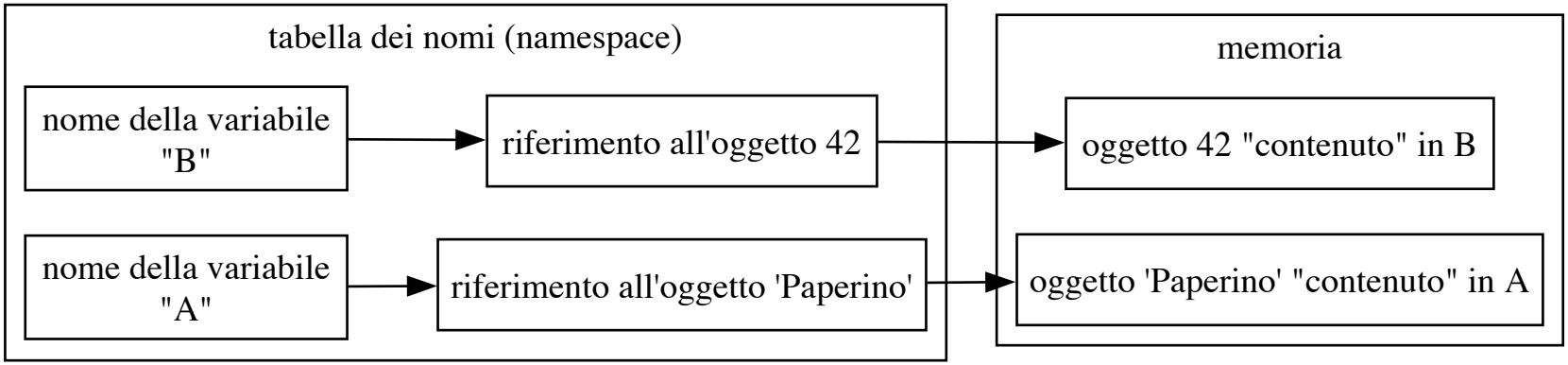
Allora perchè in Python c'è un doppio legame?

Nei linguaggi interpretati e non tipati come il Python si è scelto di permettere ad una variabile di **contenere un dato di tipo qualsiasi** anche in **momenti diversi dell'esecuzione**.

Quindi la tabella dei nomi contiene i riferimenti, che a loro volta indicano il dato (di tipo qualsiasi)

In [1…
```
figura
```

Out[1…



Come modificare una variabile?

Con l'operazione di assegnamento `=`

> `nome_variabile = espressione`

L'assegnamento cambia il **riferimento** associato al **nome** nella **tabella dei nomi (namespace)**

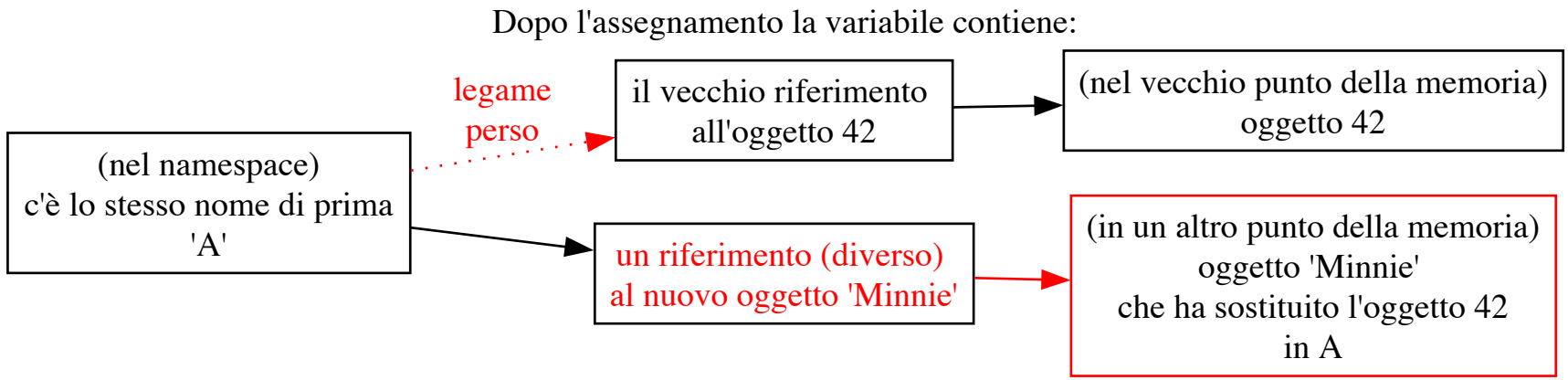
In [2…
```
from pygraphviz import AGraph
T = "Dopo l'assegnamento la variabile contiene:"
A = "(nel namespace)\nc'è lo stesso nome di prima\n'A'"
R1 = "il vecchio riferimento \nall'oggetto 42"
O1 = "(nel vecchio punto della memoria)\noggetto 42"
R2 = "un riferimento (diverso) \nal nuovo oggetto 'Minnie'"
O2 = "(in un altro punto della memoria)\noggetto 'Minnie' \nche ha sostituito l'oggetto 42\nin A"
figura3 = AGraph(directed=True, rankdir='LR', label=T, labelloc='t')
figura3.node_attr.update(shape='box')
figura3.add_node(A)
figura3.add_node(R1)
figura3.add_node(O1)
figura3.add_edge(A,R1,label='legame\nperso',fontcolor='red',color='red',style='dotted')
```

```
figura3.add_edge(R1,O1)
figura3.add_node(R2, fontcolor='red')
figura3.add_node(O2, color='red')
figura3.add_edge(A,R2)
figura3.add_edge(R2,O2, color='red')
figura3.layout('dot')
```

In [2…

```
# dopo aver assegnato un nuovo valore alla variabile A
# A = 42
# A = 'Minnie'
figura3
```

Out[2…



In [2…

```
# Come individuare il riferimento di un oggetto? con l'istruzione `id`
A = 'Paperino'
B = 'Minnie'
print('A =', A, 'B =', B)
print('id(A) =', id(A), 'id(B) =', id(B))
```

```
A = Paperino B = Minnie
id(A) = 4405292016 id(B) = 4405303296
```

In [2…

```
A = B       # se modifico la variabile A inserendoci il valore contenuto in B
print('dopo aver eseguito A=B')
print('A =', A, 'B =', B)
print('id(A) =', id(A), 'id(B) =',id(B))
# dopo l'assegnamento A contiene lo stesso riferimento di B
# quindi lo stesso oggetto in memoria è accessibile sia da A che da B
```

```
dopo aver eseguito A=B
A = Minnie B = Minnie
id(A) = 4405303296 id(B) = 4405303296
```

In [2…
```python
# due variabili "contengono" lo stesso identico oggetto SOLO se gli id sono uguali
# possiamo controllarlo con l'operatore di confronto `is`
print('A contiene lo stesso oggetto di B?', A is B)

# vediamo se confronto A con un oggetto diverso
C = 42
print('A contiene lo stesso oggetto di C?', A is C)
```

```
A contiene lo stesso oggetto di B? True
A contiene lo stesso oggetto di C? False
```

In [2…
```python
# due oggetti possono avere ID diversi ma contenere le stesse informazioni
A = 'Paperino '*4         # costruisco una stringa usando la ripetizione
B = 'Paperino '*4         # ne costruisco un'altra con lo stesso contenuto
print('A=',A)
print('B=',B)
```

```
A= Paperino Paperino Paperino Paperino
B= Paperino Paperino Paperino Paperino
```

In [2…
```python
# due oggetti sono lo stesso oggetto in memoria? uso l'operatore 'is'
print('A contiene lo stesso esatto oggetto di B?', A is B)
# due oggetti sono simili? uso l'operatore '=='   (doppio uguale)
print('A contiene le stesse informazioni di B?', A == B)
```

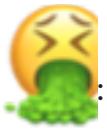
```
A contiene lo stesso esatto oggetto di B? False
A contiene le stesse informazioni di B? True
```

Questioni di stile: come scegliere i nomi delle variabili

Scegliete un nome **PARLANTE** che descrive il dato contenuto

- è più facile da ricordare
- con l'autocompletamento del codice degli IDE i nomi lunghi non sono faticosi da scrivere
- il codice diventa più discorsivo e leggibile/comprensibile

Esempi:

- **GOOD**: `elenco_altezze` (descrive il dato)
- **MEH**: `lista` (non dice nulla del tipo di elementi contenuto nella lista)
-
-  : `L` (non dice nulla nè del dato nè degli elementi)

Conversioni tra tipi diversi

- si usa il nome del tipo `tipo(valore da convertire)`

In [2…
```
# da numeri a stringhe
# (voglio il testo che li rappresenta stampandoli come sequenza di caratteri)
str(42), str(17.34)
```

Out[2…
```
('42', '17.34')
```

In [2…
```
# da stringhe a numeri int o float
# voglio il numero partendo dalla sua rappresentazione testuale
int('23'), int('-46'), float('23'), float('-46'), float('3.2415')
```

Out[2…
```
(23, -46, 23.0, -46.0, 3.2415)
```

In [2…
```
# ATTENZIONE!
# che succede se la stringa non rappresenta un numero di quel tipo?
int('3.5') # chiedo un intero ma la stringa non è un intero

# ERRORE!!!! "Tradotto: il testo non rappresenta un intero in base 10"
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[29], line 3
      1 # ATTENZIONE!
      2 # che succede se la stringa non rappresenta un numero di quel tipo?
----> 3 int('3.5') # chiedo un intero ma la stringa non è un intero
      5 # ERRORE!!!! "Tradotto: il testo non rappresenta un intero in base 10"

ValueError: invalid literal for int() with base 10: '3.5'
```

In [3…
```
int('ABC')    # Errore: Tradotto: "in base 10 questo non è un numero"
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[30], line 1
----> 1 int('ABC')    # Errore: Tradotto: "in base 10 questo non è un numero"

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

In [3…
```
# ma se fornisco anche la base del numero contenuto nella stringa...
print(int('ABC',16))
int('C', 16)   # in base 16 i numeri dopo il 9 sono rappresentati
               # dalle lettere A=10, B=11, C=12, D=13, E=14 e F=15
```

```
2748
```

Out[3… 12

Come leggere un dato con `input`

```
testo = input("Messaggio che chiede di inserire il dato (prompt)")
```

NOTA: il risultato è SEMPRE una stringa (che potete poi convertire in intero o float)

In [3…
```
# input di un testo
nome = input("Come ti chiami? ")
print("Mi chiamo", nome)
```

```
Mi chiamo Andrea
```

In […
```python
# input di un intero
testo = input("Quanto sei alto? (intero) ")
altezza = int(testo)
print("Sono alto ", altezza, "cm")
```

In [3…
```python
# input di un float
testo = input("Quanto pesi?")
peso = float(testo)
print("Peso ", peso, "kg")
```

```
Peso  98.5 kg
```

Ancora a proposito di stringhe

Le **stringhe** `str` sono piu complesse di `int` e `float`.
E' possibile pensarle codificate in memoria come `array` (sequenze di elementi uguali).

In [8…
```python
nome = 'Andrea Sterbini'
```

carattere	**A**	**n**	**d**	**r**	**e**	**a**		**S**	**t**	**e**	**r**	**b**	**i**	**n**	**i**
indice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

1. Si ottiene la **lunghezza** della sequenza di caratteri con **`len(nome)`** (15 in questo caso)

2. E' possibile anche:
   - indicizzare la stringa, ossia accedere **in maniera secca** (`random access`) a ciascun carattere
   - per vedere il carattere quinto, non è necessario che esamini primo, secondo, terzo e quarto. Accedo direttamente al quinto **`nome[4]`**

3. Molto importante!

   in informatica si inizia a contare da 0

   - `nome[0]` è il primo elemento

carattere	A	n	d	r	e	a		S	t	e	r	b	i	n	i
indice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

In [8…
```
nome[0]
```

Out[8…
```
'A'
```

In [3…
```
# se l'indice sfora ottengo un errore di tipo IndexError
nome[100]  #attenzione il mio nome e cognome non arriva a 100 caratteri
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[37], line 2
      1 # se l'indice sfora ottengo un errore di tipo IndexError
----> 2 nome[100]  #attenzione il mio nome e cognome non arriva a 100 caratteri

IndexError: string index out of range
```

In [8…
```
len(nome) #infatti i caratteri sono 15
```

Out[8…
```
15
```

In [8…
```
# ok accediamo al quindicesimo
nome[15]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[86], line 2
      1 # ok accediamo al quindicesimo
----> 2 nome[15]

IndexError: string index out of range
```

😨

carattere	A	n	d	r	e	a		S	t	e	r	b	i	n	i
indice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

In [8…
```
start = nome[0]            # leggo il primo elemento, lo metto dentro la variabile start
end = nome[len(nome)-1]    # calcolo la lunghezza della stringa.
                           # ultimo elemento è in posizione luneghezza-1.
                           # lo leggo e lo metto dentro end
print(start)  #stampa la prima lettera
print(end)    #stampa l'ultima lettera
```

```
A
i
```

In [8…
```
# Posso anche a contare a partire dalla fine
ultimo = nome[-1]
penultimo = nome[-2]
print(ultimo)
print(penultimo)
```

```
i
n
```

Slicing (affettamento) di sequenze

Abbiamo visto che con la sintassi `stringa[indice]` possiamo estrarre un elemento

Possiamo anche usare la sintassi `stringa[inizio:fine]` per estrarre un gruppo di elementi che:

- inizia all'indice `inizio`
- termina SUBITO PRIMA dell'indice `fine`

In [8…
```
# possiamo anche "affettare" la stringa (slicing)
# per ottenerne un segmento
# ed estrarre ad esempio il mio nome
```

```
nome[0:6] # da notare MOLTO IMPORTANTE
          # 0 e' compreso, 6 escluso
```

Out[8… 'Andrea'

car.	A	n	d	r	e	a
idx	0	1	2	3	4	5

Assumiamo di volere estrarre `drea`
da `Andrea Sterbini`.

- Come facciamo?
- Ricordiamoci che il carattere spazio nel mezzo **e'** un carattere (lo spazio si codifica nel computer)

carattere	A	n	d	r	e	a		A	t	e	r	b	i	n	i
indice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

- Possiamo fare slicing con l'operatore `:` da 2 a 6 ossia `[2:6]`
  (prendi la fetta che va dall'indice 2 incluso al 6 escluso)

- Ricordatevi `gli indici di slicing` in python sono sempre intervalli $[\text{ start},\text{end })$
- `start` incluso, `end` **ESCLUSO**

In [9…
```
sub_name = nome[2:6]
print(sub_name)
```

```
drea
```

E se non sappiamo dove inizia `drea`, che facciamo?

- Sappiamo che dobbiamo cercare `drea`
- `drea` è lungo 4 caratteri, ma possiamo usare il metodo `len()` per calcolarsi la lunghezza della stringa, cosi funzionerà con tutte le stringhe, non solo con `drea`.
- Come facciamo a trovare l'indice di inizio di `drea` dentro `nome`?

> Se ci fate caso ci stiamo piano piano approcciando al **problem solving**

In [5…
```
# help(str) # conviene leggersi le funzionalita' del tipo str

# Tra i metodi di str c'è index
```

```
|  index(...)
|      S.index(sub[, start[, end]]) -> int
|
|      Return the lowest index in S where substring sub is found,
|      such that sub is contained within S[start:end].  Optional
|      arguments start and end are interpreted as in slice notation.
|
|      Raises ValueError when the substring is not found.
```

`S.index(sottostringa) -> int`

1. prendo la stringa `S` ci applico la funzionalita' `index` che prende `sottostringa` come argomento
2. Rende un intero `-> int` che indica la posizione trovata (oppure lancia un errore)
3. Questa e' documentazione non codice

In [9…
```
# quindi possiamo fare
# nome vale 'andrea sterbini '
start = nome.index('drea')
start

#nome.find('pippo')
```

Out[9… 2

In [9…
```
nome[start:start+len('drea')] # [2,2+4)
```

Out[9… 'drea'

d	r	e	a
2	3	4	5

In [9…
```
# mettendo tutto insieme
query = 'drea'                    # input dell'algoritmo
start = nome.index(query)         # trovo il primo indice che contiene il contenut di QUERY
                                  # sulla contenuto di NOME
end = start + len(query)          # mi precalcolo indice di fine stringa da copiare
sub_name = nome[start:end]        # faccio slicing della stringa
print(sub_name,start,end,nome, sep='\n')    # stampo cosa ho trovato e dove, a partire da cosa
```

```
drea
2
6
Andrea Sterbini
```

d	r	e	a
2	3	4	5

Di più su slicing

- E' possibile anche **non mettere il valore di start e end**
  - per default si tratta dell'inizio o della fine
- E' possibile **saltare ogni K elementi**
  - indicando un terzo parametro, l'incremento
- E' possibile andare **al contrario**
  - indicando un incremento negativo

In [9…
```
# voglio tutti i caratteri DOPO il
# primo (indice 0 escluso, 1 compreso)
nome[1:]
```

Out[9…
```
'ndrea Sterbini'
```

In [9…
```
nome[:2] # tutti i caratteri PRIMA
         # del indice 2 escluso
```

Out[9…
```
'An'
```

In [9…
```
nome[::2]
```

Out[9…
```
'Ade trii'
```

In [9…
```
nome[-1]  # al contrario di molti linguaggio a basso livello
          # qui gli indici possono essere negativi
          # il meno inverte la sequenza di lettura
          # sto prendendo da 'andrea sterbini' <--- questa i finale
```

Out[9…
```
'i'
```

- Questa complessità è molto utile e efficace ma ovviamente **va saputa gestire**
- Abbastanza facile ingarbugarsi su `slicing` complessi

In [9…
```
nome = 'Python'
vuoto = nome[4:3] # ovviamente se start >= end, abbiamo stringa vuota
type(vuoto),vuoto
```

Out[9…
```
(str, '')
```

In [1…
```
nome = 'Python'
vuoto = nome[-3:-4] # ovviamente se start >= end, abbiamo stringa vuota ''
type(vuoto),vuoto
```

Out[1…
```
(str, '')
```

In [1…
```
nome = 'Python'
nome[-4:-3] == nome[2:3], nome[2:3]
# questo funzione seleziona il carattere 't' (un singolo carattere e' una stringa)
```

Out[1…
```
(True, 't')
```

Index ->	0	1	2	3	4	5	
"	P	y	t	h	o	n	"
	-6	-5	-4	-3	-2	-1	<- Reverse index

NOTA:

- l'indice (da sinistra a destra) va da **0** a **len(stringa)-1**
- l'indice inverso (da destra a sinistra) va da **-1** a **-len(stringa)**

In [1…
```
nome = 'Python'
nome[-10:2], nome[3:100]
# se l'inizio è negativo e troppo grande si inizia dal primo carattere
# se la fine della slice è troppo grande si arriva all'ultimo
```

Out[1…
```
('Py', 'hon')
```

In [1…
```
print(nome[-len(nome)])
nome[-7]
# MA ATTENZIONE: se invece indicizziamo il **singolo carattere**
# non possiamo sforare e chiedere più di -(len(nome))
```

P

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[103], line 2
      1 print(nome[-len(nome)])
----> 2 nome[-7]
      3 # MA ATTENZIONE: se invece indicizziamo il **singolo carattere**
      4 # non possiamo sforare e chiedere più di -(len(nome))

IndexError: string index out of range
```

In [1…
```
# Prendiamo tutti i caratteri in posizione multipla di 3
# il terzo valore indica come incrementare l'indice
nome[::3]
```

Out[1…
```
'Ph'
```

In [1…
```
# prendiamoli in direzione opposta (incremento negativo)
nome[::-1]
```

Out[1…
```
'nohtyP'
```

Momento Wooclap

FDPLEZ2

quanto fa: `'ippopotamo'[2:-2:2]` ?

e invece `'BabbuINO'.lower()[-1:-10:-3]` ?

In [1…
```
# Risultati
'ippopotamo'[2:-2:2], 'BabbuINO'.lower()[-1:-10:-3]

# RICORDATE: l'indice nel secondo parametro NON viene raggiunto
# infatti 'm' non è presente nella prima risposta
```

Out[1…
```
('ppt', 'oua')
```

Stringhe come Tipo Immutabile

- I tipi **mutabili e immutabili** in python li vediamo meglio nelle prossime lezioni
- Informalmente, un tipo **immutabile** vuol dire che una volta che e' stato creato **NON** può essere più modificato
- Per chi viene dal `C` questa cosa non e' molto chiara perchè in C potete modificare direttamente le stringhe carattere per carattere

In [1…
```
nome = 'Python'
```

assumiamo io voglia modificare la `P` di `Python` in minuscola `python`

Per modificare un elemento in una certa posizione di una stringa?

Potrei usare la stessa notazione che ho usato per leggerlo

`nome[indice] = carattere`

In [1…

```
# in C potrei sostituire il primo carattere con 'p' minuscola
nome[0] = 'p'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[108], line 2
      1 # in C potrei sostituire il primo carattere con 'p' minuscola
----> 2 nome[0] = 'p'

TypeError: 'str' object does not support item assignment
```

ERRORE: (tradotto) **le stringhe non permettono la modifica di un loro elemento**

In realta' non lo posso fare direttamente, ma devo per forza creare un'altra stringa

NOTA: Altri tipi di contenitori sono invece modificabili e permettono di inserire un nuovo valore in una posizione

In [1…

```
# Esempio: voglio la stringa in minuscole o MAIUSCOLE
A = nome.lower()
print(A)
print(A.upper())
```

```
python
PYTHON
```

In [1…
```
# oppure potrei fare cut/paste usando slice e concatenazione
B = 'p'+nome[1:]
B
```

Out[1…
```
'python'
```

In [1…
```
# con id(oggetto) oppure con 'is' posso capire se due cose sono lo stesso oggetto in memoria
id(A), id(B), id(A)==id(B), A is B
# le due stringhe sono in posti uguali della memoria ? (NO)
```

Out[1…
```
(4405335232, 4405007600, False, False)
```

In [1…
```
help(id)
```

```
Help on built-in function id in module builtins:

id(obj, /)
    Return the identity of an object.

    This is guaranteed to be unique among simultaneously existing objects.
    (CPython uses the object's memory address.)
```

String Literals

Se dobbiamo scrivere testi che contengono accapi (`'\n'`) possiamo usare il triplo apice `'''` oppure il triplo doppio apice `"""`

In [1…
```
print("""
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
""")
```

```
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
```

String interpolation (ESTREMAMENTE UTILE!!!)

Possiamo costruire facilmente dei testi che contengono valori calcolati (ad es. presi da variabili o calcolati con espressioni)

- precedendo la stringa con la lettera **`f`** (che sta per 'formatted')
- inserendo i valori da interpolare tra parentesi graffe **`{espressione}`** oppure **`{variabile}`**
- se vogliamo, indicando come visualizzare il dato interpolato (vedi documentazione)
  - aggiungendo nelle graffe, dopo l'espressione **`:<formato>`** esempio:
    - `:.2f` float con due cifre decimali
    - `:%` percentuale
    - `:>30s` stringa appoggiata a destra in uno spazio di 30 caratteri

In [1…

```
# Esempio: definiamo le parti da interpolare in un invito da mandare per email
nome      = 'Paolino'
cognome   = 'Paperino'
indirizzo = 'Via dei Peri 32'
data      = '29 febbraio'
orario    = '20:30'
mittente  = 'Gastone Paperone'
```

In [1…

```
# e il testo da spedire (che interpola semplici variabili)
lettera = f'''
Caro {nome.upper()} {cognome}
La invito al vernissage che si terrà a {indirizzo}
il giorno {data} alle ore {orario}
Cordialmente
{mittente}
'''
```

In [1…

```
print(lettera)
```

```
Caro PAOLINO Paperino
La invito al vernissage che si terrà a Via dei Peri 32
il giorno 29 febbraio alle ore 20:30
Cordialmente
Gastone Paperone
```

Valori vero/falso (di tipo `bool`)

Valori che indicano se il risultato di un confronto è vero o falso

- **`True`**: valore vero
- **`False`**: valore falso

Vengono usati nelle condizioni delle istruzioni di controllo che servono ad indicare al programma di eseguire parti diverse a seconda se la condizione è vera o falsa

NOTA: Sono unici in tutto il sistema, quindi facili da verificare

In [1…
```python
# Per comodità di programmazione, se interpretati come bool,
# alcuni valori valgono/significano "VERO", altri "FALSO"
# in pratica si vuol distinguere lo 0 o gli oggetti vuoti dal resto

# il valore intero 0, se lo consideriamo come bool è False,
# tutti gli altri interi sono True
bool(0), bool(1), bool(2), bool(-42)
```

Out[1…
```
(False, True, True, True)
```

In [1…
```python
# che tipo hanno i due valori?
type(True), type(False)
```

Out[1…
```
(bool, bool)
```

In [1…
```python
# Anche per i float, 0.0 vale False mentre tutti gli altri float valgono True
bool(0.0), bool(0.00000001),  bool(0.5), bool(-3.14)
```

Out[1… (False, True, True, True)

In [1…
```
# viceversa, se proviamo a trasformare True/False in int o float
# i due valori True e False corrispondono
# agli interi 1 e 0   e ai float 1.0 e 0.0
int(True), int(False), float(True), float(False)
```

Out[1… (1, 0, 1.0, 0.0)

Operatori di confronto tra valori

Come confrontare oggetti?	Operatore
Sono esattamente lo stesso oggetto in memoria?	`is`
Contengono le stesse informazioni? sono simili?	`==`
Il primo è strettamente minore? (sta prima nell'ordinamento)	`<`
Il primo è strettamente maggiore? (sta dopo nell'ordinamento)	`>`
Il primo è minore o uguale?	`<=`
Il primo è maggiore o uguale?	`>=`

In [1…
```
# Posso confrontare numeri (anche di tipo diverso)
print('12 < 11.3?', 12 < 11.3)
```

```
12 < 11.3? False
```

In [1…
```
# Posso confrontare stringhe rispetto all'ordinamento alfabetico
print("'Paperino' < 'Topolino'?", 'Paperino' < 'Topolino')
```

```
'Paperino' < 'Topolino'? True
```

In [1…
```
# MA NON POSSO confrontare numeri e stringhe (sono due ordinamenti separati)
print("42.17 < 'Topolino'?", 42.17 < 'Topolino')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[123], line 2
      1 # MA NON POSSO confrontare numeri e stringhe (sono due ordinamenti separati)
----> 2 print("42.17 < 'Topolino'?", 42.17 < 'Topolino')

TypeError: '<' not supported between instances of 'float' and 'str'
```

In [1…
```python
# A meno di non convertire prima il numero in una stringa
print("str(42.17)='42.17' < 'Topolino'?", str(42.17) < 'Topolino')
```

```
str(42.17)='42.17' < 'Topolino'? True
```

Come combinare valori logici con gli operatori `and`, `or` e `not`

Espressioni logiche:

- `espressione` **`and`** `espressione` : è vera solo se entrambe le espressioni sono vere
- `espressione` **`or`** `espressione` : è vera solo se almeno una è vera
- **`not`** `espressione` : rovescia True <-> False

L'ordine di precedenza è: **`not`** -> **`and`** -> **`or`**

In [1…
```python
# Esempio
ho_portato_ombrello = False
piove = True
mi_bagno = not ho_portato_ombrello and piove    # ovvero (not ho_portato_ombrello) and piove
print("Oggi mi bagno? ", mi_bagno)
```

```
Oggi mi bagno?  True
```

Momento Wooclap

FDPLEZ2

quanto fa: `True and not False or True and False or not True` ?

e invece `True+False/True+True*True-False` ?