UNIVERSITÀ DEGLI STUDI DELLA TUSCIA

PROGRAMMAZIONE

Linguaggio Python
Variabili e set istruzioni

Dott. Franco Liberati
franco.liberati@unitus.it

LINGUAGGIO PYTHON

Argomenti del corso

PYTHON

OGGETTO: definizione

- L’entità atomica che il Python crea e manipola è definita **oggetto**
- Per **oggetto** si intende un contenitore (una porzione di memoria) nel quale è stipato un valore a cui è associata una o più variabili o una funzione o altro ancora (la logica è la stessa del linguaggio R)

PYTHON

VARIABILE: identificatore

- Ogni oggetto-Python è definito da un **identificatore** cioè una stringa alfanumerica che deve iniziare con un carattere (mai una cifra) e può sfruttare il carattere underscore (_) e il punto (.); dopo il punto non può esserci un numero. Esistono poi parole riservate (if, while,...) che non possono essere utilizzate dal programmatore

Oggetto-Python valido

i
__mio_nome,
nome_23
a1b2_c3

Oggetto-Python NON valido

2cose
questo è spaziato
mio-nome
"questo è tra virgolette"

PYTHON

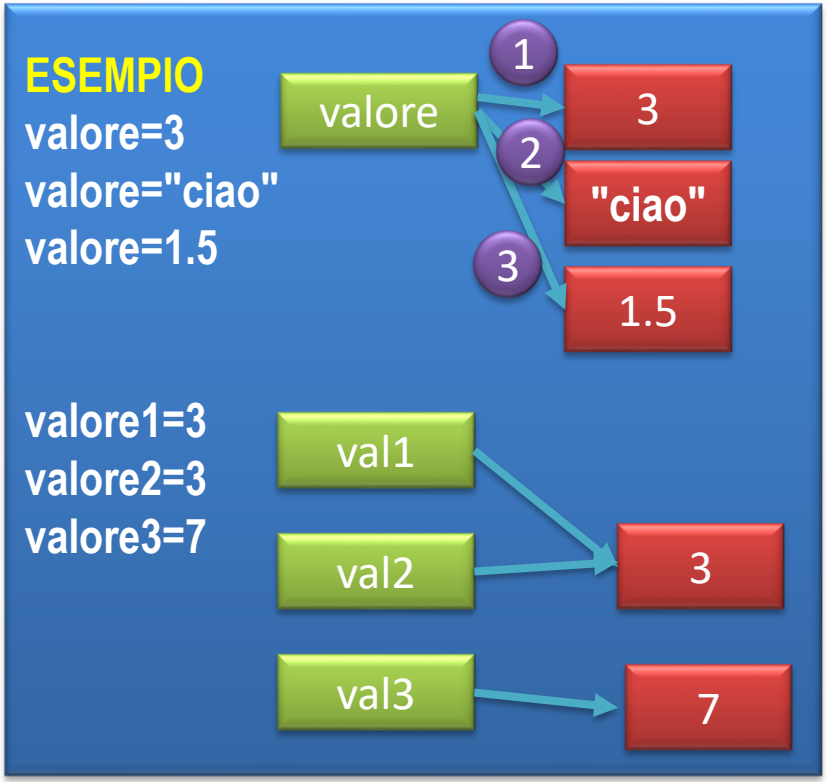
VARIABILE: identificatore

- Una variabile è creata quando le si assegna un valore
- Un oggetto ha associato un tipo (e non alla variabile)
- Quando una variabile appare in un espressione viene rimpiazzata dall'oggetto associato

PYTHON

VARIABILE: identificatore

- Ciascun oggetto possiede un **type designator** e un **reference counter**
- La variabile non ha un tipo associato. Cambia solo la il riferimento (reference)
  Il tipo 'vive' con l'oggetto non con la variabile (per questo di parla di type designator)
- Ciascun oggetto ha associato un reference counter, un contatore al numero di riferimenti (reference) associate all'oggetto.



PYTHON

TIPI

- Python dispone di **due tipologie di dato**
  - DATI SEMPLICI
    - Int
    - Long
    - Float
    - Complex
    - String
  - CONTENITORI
    - tuple ()
    - list []
    - dict {}
    - set

PYTHON

TIPO SEMPLICE: int

- **Integer**
  - Il tipo **int** è valido per tutti i numeri interi che sono compresi tra [-2147483648 e 2147483647], vedi (sys.maxint)
  - Un intero può essere espresso in base decimale oppure in base esadecimale o ottale antecedendo al numero rispettivamente **0x** e **0**.

ESEMPI INT

```
a=300              #decimale
b=0x12c            #esadecimale
c=0454             #ottale
a_oct=oct(a)       #ottale
a_hex=hex(a)       #esadecimale
```

PYTHON

TIPO SEMPLICE: long

- **Long Integer**
  - Il tipo **long** è analogo al tipo intero con l'unica eccezione che il valore massimo e minimo che può assumere è limitato solo dalla memoria a disposizione. La sua definizione richiede la lettera **L** dopo il numero

ESEMPI LONG

```
a = 1254546699L
b = 484564848766
#print(b) da come risultato 484564848766L
e=2**1024
#print (e) da come risultato
179769313486231590772930519078902473361797697894230657273430081157732675805500963132708477322407536021120113879871393357658789768814416622492847430639474124377767893424865485276302219601246094119453082952085005768838150682342462881473913110540827237163350510684586298239947245938479716304835356329624224137216L
```

PYTHON

TIPO SEMPLICE: float

- **Floating Point Number**
  - Il tipo **float** rappresenta numeri reali in doppia precisione

ESEMPI FLOAT

a = 12.456
c = 12232e-2
b = 0.2
d = 6.12244e-5

In Python se assegno a=0.2 e poi lo stampo print(a) si ottiene 0.2000000000000001
La rappresentazione dei numeri è binaria. Alcuni numeri decimali non possono essere rappresentati correttamente e questo provoca piccoli errori di arrotondamento

Il prompt di Python di default utilizza la funzione **repr()** per il display di numeri floating point, che per default approssima alla 17-esima cifra significativa. Quando si necessita maggiore accuratezza (p.e. applicazioni finanziarie) è possibile ricorrere all'utilizzo del modulo **Decimal**: la precisione è specificata dall'utente e i numeri frazionari binari sono rappresentati esattamente a scapito di un più lento processamento dei dati.

PYTHON

TIPO SEMPLICE: complex

- **Complex Number**
  - Un numero **complex** rappresenta un tipo numerico complesso in doppia precisione
  - Si accede alla parte reale e immaginaria di un numero complesso attraverso le funzioni ‘real’ e ‘imag’

ESEMPI COMPLEX

```
r=12+5j
print(r.real)
12
print(r.imag)
5.0
```

NB: ‘j’ indica la parte immaginaria

PYTHON

TIPO SEMPLICE: boolean

- **BOOLEAN**
  - È possibile utilizzare i **valori interi** per rappresentare valori booleani con la convenzione che 0 corrisponda a **False** e tutti i valori interi positivi corrispondano a **True**
  - È tuttavia buona norma di programmazione utilizzare il tipo **bool** per rappresentare valori booleani TRUE e FALSE
    - Una variabile di tipo bool può assumere valori TRUE e FALSE che sono interscambiati con i valori 1 e 0

ESEMPI BOOLEAN

```
a=1
type(a) <type 'int'>
if(a):
        print 'True'
True

_____

a=False
type(a) <type 'bool'>
```

PYTHON

TIPO SEMPLICE: string

- **STRING**
  - Una **stringa è una sequenza di caratteri racchiusa tra doppi o singoli apici**
  - Le sequenze di apici tripli """ o ''' possono essere utilizzate per stringhe che spaziano su più righe, o che contengono apici singoli o doppi (o tripli dell'altro tipo)

ESEMPI STRING

```
a='ciao’
b="Mondo"
```

ESEMPI STRING

```
a=""" Sono una stringa e contengo apici 'singoli', "doppi" e "'tripli'" """
print (a)
 Sono una stringa e contengo apici 'singoli', "doppi" e "'tripli'"
```

PYTHON

TIPO SEMPLICE: string

- **STRING**
  - Una stringa è una sequenza di caratteri: lettere, numeri, simboli e sequenze di escape (o di controllo, es:\t).
    L'escaping (*simbolo*) permette di effettuare il quoting di un singolo carattere
    Attraverso l'escaping è possibile aggiungere apici o altri caratteri all'interno di una stringa.

ESEMPIO ESCAPING

```
a = 'C'era una volta'    #Errore
SyntaxError: invalid syntax
a = "C'era una volta"    # Ok con doppio apice
a = 'C\'era una volta'    # Ok con escape sentence, cioè \'
```

PYTHON

TIPO SEMPLICE: string (assegnazione

- Per accedere al singolo carattere si può ricorrere all'operatore *stringa*[*indice*] oppure ad una sottostringa con l'operatore [begin:end] (*operatore di slicing*)

NB: il valore *end* è escuso

ESEMPIO

```
a = "Hello world"
lettera=a[1]
print(lettera)
'e'
lettere=a[0:4]
print(lettere)
'Hell'
```

- Non è possibile modificare il singolo carattere, ma è possibile assegnare alla stringa un nuovo valore

ESEMPIO

```
a="Primo valore"
a = "Cambio"
a[2] = 't' #NO!!!!!!
TypeError: 'str' object does not support item assignment
```

PYTHON

TIPO SEMPLICE: string (caratteri speciali con escape)

- Nelle stringhe si usano caratteri speciali
  - \t Tabulatore
  - \n Andata a capo
  - \\ Backslash
  - \" Doppio apice
  - \' singolo apice
  - \b Cancellazione di un carattere

ESEMPIO

```
a="Ciao\tciao!"
print(a)
```
Ciao ciao!

```
a="Ciao\nciao!"
print(a)
```
Ciao
ciao!

```
a="C:\\\\Programmi\\Python\\Python.exe"
print(a)
```
C:\\Programmi\Python\Python.exe

```
a="Ciao a \"TUTTI\" i ragazzi della Tuscia"
print(a)
```
Ciao a "TUTTI " i ragazzi della Tuscia

```
a="Ciao ai ragazzi\b della Tuscia"
print(a)
```
Ciao a i ragazz della Tuscia

PYTHON

TIPI: riconoscimento

- Per conoscere il tipo di oggetto associato ad una variabile si utilizza la funzione built-in **type(**oggetto**)**

ESEMPIO

```
a=2147483647
type(a)
<type 'int'>
a=a+1
type(a)
<type 'long’>
a=13.23
type(a)
<type ‘float’>
a=“Ciao”
type(a)
<type ‘str'>
```

Set istruzioni

PYTHON

SET ISTRUZIONI : interazioni con dispositivi I/O

- Per leggere un oggetto (numero o stringa) da tastiera si può ricorrere alla funzione

variabile=input(*prompt*)

prompt è un messaggio da inviare all'utente su videoterminale

Il contenuto letto è una stringa

ESEMPI

```
val1 = input("Inserisci una stringa: ")
print(val1)
#Con input Pippo
Pippo
print(type(val1))
<class 'str'>
```

```
val2 = input("Inserisci un numero: ")
print(val2)
#Con input 45
45
print(type(val2))
<class 'str'>
val3=int(val2)
print(val3)
45
print(type(val3))
<class 'int'>
```

PYTHON

SET ISTRUZIONI : interazioni con dispositivi I/O

- Per **stampare un oggetto** (numero o stringa) da tastiera si può ricorrere alla funzione

print(*object(s)*, sep=*separator*, end=*end*, file=*file*, flush=*flush***)**

ESEMPI

```
print("Ciao!", "Come stai?")
```

Ciao! Come stai?

```
val="Franco"
print("Il tuo nome ?", val)
```

Il tuo nome è Franco

```
separatore="\t"
print("Marco", "Giovanni", sep=separatore)
```

Marco Giovanni

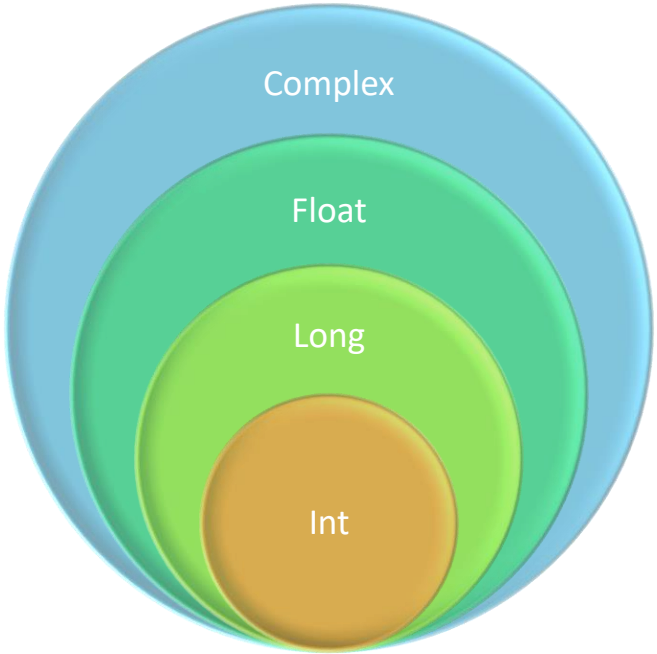
```
separatore="-"
print("Marco", "Giovanni", sep=separatore)
```

Marco-Giovanni

PYTHON

SET ISTRUZIONI : aritmetiche

Nello svolgere operazioni tra variabili numeriche di diverso tipo è seguita la seguente regola di conversione implicita (es.: se moltiplico un intero con un float il risultato ha tipo float



PYTHON

SET ISTRUZIONI : aritmetiche

Simbolo	Significato	Esempio
+	Addizione	x=56; y=77 som=x+y print(som) **133**
-	Sottrazione	sot=x-y print(sot) **-21**
*	Moltiplicazione	mol=x*y print(mol) **4312**
/	Divisione	div=x/y print(div) **0.7272727272727273**
//	Quoziente della divisione	div_i=x//y print(div_i) **0**
%	Resto della divisione	res=x%y print(res) **56**
**	Elevamento a potenza	b=2; e=3; pow=b**e print(pow) **8**

PYTHON

SET ISTRUZIONI : aritmetiche (funzioni ausiliarie)

Funzione	Significato	Esempio
abs(*val*)	Modulo	z=-15 print(abs(z)) **15**
round(number[, ndigits])	Arrotondamento	t=9.87678765 print(round(t,4)) **9.8768**
new_type(*val*)	Cambio di tipo (*coerce*) Dove new_type è: int float complex bool str ord	x_int = 19 x_float =float(x_int) print(x_float) **19.0** print(type(x_float)) **<class 'float'>**

PYTHON

SET ISTRUZIONI : logici bit a bit

Simbolo	Significato	Esempio
&	AND	x=56 (cioè 0011 1000) y=77 (cioè 0100 1101) andlogico=x&y print(andlogico) 8 (cioè **0000 1000**)
|	OR	x=56; y=77 orlogico=x|y print(orlogico) 125 (cioè **0111 1101**)
~	NOT	x=56; (cioè 0011 1000) notlogico=~x print(notlogico) -57 (cioè **1100 0111**)

PYTHON

SET ISTRUZIONI : operatori logici

Simbolo	Significato	Esempio
and	And logico	a=True b=False c=a and b print (c) **False**
or	Or logico	a=True b=False c=a or b print (c) **True**
not	Not logico	a=True b=False c=not a print (c) **False** d=not b print (d) **True**

PYTHON

SET ISTRUZIONI : operatori di confronto

Simbolo	Significato	Esempio
<	Minore *(ordine lessicografico)*	3<5 **TRUE**
<=	Minore uguale *(ordine lessicografico)*	5<=5 **TRUE**
==	Uguale	4==7 **FALSE**
!=	Diverso	4!=4 **FALSE**
>	Maggiore *(ordine lessicografico)*	5>3 **TRUE**
>=	Maggiore o uguale *(ordine lessicografico)*	6>=6 **TRUE**

PYTHON

SET ISTRUZIONI : il modulo MATH

- Il modulo **math** fornisce alcune delle più comuni funzioni matematiche

- Le funzioni disponibili sono:
  - Funzioni **trigonometriche**: cos, sin, tan, asin, acos, atan, sinh, cosh, tanh.
  - Funzioni di **elevamento potenza** : pow, exp,
  - Funzioni **logaritmo:** log, log10
  - **Radice quadrata**: sqrt
  - Funzioni di **Rappresentazione:** ceil , floor, fabs,
  - Funzioni di T**rasformazione angoli:** degrees, radians.

- Nel modulo math sono inoltre definite le costanti numeriche **pi** (pi-greco) ed **e** (numero di Nepero)

PYTHON

SET ISTRUZIONI : il modulo MATH

- Per utilizzare il modulo math è sufficiente importarlo con la seguente sintassi:

  import math

- Si possono importare tutte le funzioni con la sintassi

  from math import *

- Si possono importare solo alcune funzioni con la sintassi

  from math import sin, cos (import specific function sinx, cosx)

- Una volta importate si può avere la lista delle funzioni mediante il comando

  dir(math)

['__doc__', '__name__', '__package__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'hypot', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']

PYTHON

SET ISTRUZIONI : stringhe

Simbolo	Significato	Esempio
+	Concatenazione	a = 'Ciao' concatenazione=a+a+a print (concatenazione) **'CiaoCiaoCiao'**
*	Ripetizione	concatenazione_ripetizione = 'Dol'+'c'*5+'i'+'s'*5+'imo cioccolatino' print(concatenazione_ripetizione) '**Dolccccciisssssimo cioccolatino**'
%	Stampa	
	%s specifica una stringa %c specifica un carattere %d specifica un intero %f specifica un reale (float) %g specifica un reale in formato scientifico (formato esponenziale)	a="Oggi è %s %d %s" % ("Lunedì",25,"Dicembre") print(a) **"Oggi è Lunedì 25 Dicemebre"**

PYTHON

SET ISTRUZIONI : stringhe

Simbolo	Significato	Esempio
split([sep [,maxsplit]])	Separazione in relazione a un separatore (con specificato il numero di separazione da svolgere (se omesso si applica a tutte le occorrenze del separatore)	s='Ciao Mondo' s.split('o',1) **["Cia", " Mondo"]** s='Ciao a tutte le Tuscine' s.split(' ') **["Ciao", "a", "tutte", "le", "Tuscine"]**
replace (old, new[, count])	Sostituzione di una sottostringa in una stringa	s='Ciao Mondo' s.replace('o','i',1) **'Ciai Mondo'**
strip([chars])	Eliminazione di caratteri	s='Ciao Mondo' s.strip('C') **'iao Mondo'**

PYTHON

SET ISTRUZIONI : stringhe

Simbolo	Significato	Esempio
upper()	Trasforma tutte le lettere minuscole di una stringa nelle corrispondenti lettere maiuscole	s="Ciao Tuscia" s.upper() **"CIAO TUSCIA"**
lower()	Trasforma tutte le lettere maiuscole di una stringa nelle corrispondenti lettere minuscole	s="Ciao Tuscia" s.lower() **"ciao tuscia"**
swapcase()	Trasforma tutte le lettere minuscole di una stringa nelle corrispondenti lettere maiuscole e viceversa	s="Ciao Tuscia" s.swapcase() **"cIAO tUSCIA"**

PYTHON

SET ISTRUZIONI : stringhe

Simbolo	Significato	Esempio
find(sub [,start [,end]])	Trova la posizione della prima sottostringa *sub* (si possono dare gli stremi di ricerca)	s= "Hello, welcome to my world" x = s.find("welcome") print(x) **7**
rfind(sub [,start [,end]])	Trova la posizione della ultima sottostringa *sub* (si possono dare gli stremi di ricerca)	txt = "Mi casa, est su casa." x = txt.rfind("casa") print(x) **16**
count(sub[, start[, end]])	Trova il numero di occorrenze di *sub*	txt = "I love apples, apple are my favorite fruit" x = txt.count("apple") print(x) **2**

PYTHON

SET ISTRUZIONI : stringhe

Simbolo	Significato	Esempio
<	Minore *(ordine lessicografico)*	a="Ciao" b="Mondo" print(a<b) **TRUE** a="CIAO" b="ciao" print(a<b) **TRUE**
<=	Minore uguale *(ordine lessicografico)*	
==	Uguale	a,b="Ciao","Ciao" print(a==b) **TRUE**
!=	Diverso	
>	Maggiore *(ordine lessicografico)*	
>=	Maggiore o uguale *(ordine lessicografico)*	

PYTHON

SET ISTRUZIONI : stringhe

Simbolo	Significato	Esempio
and	AND logico La stringa vuota è considerata FALSE tutte le altre TRUE	a="Ciao" c="" b="Mondo“ d=“pippo" print(a and b) print(c and d) **TRUE** **FALSE**
or	OR logico La stringa vuota è considerata FALSE tutte le altre TRUE	a="Ciao" c="" b="Mondo“ d="" print(a and b) print(c and d) **TRUE** **FALSE**
not	NOT logico La stringa vuota è considerata FALSE tutte le altre TRUE	a="Ciao" c="" print(not a) print(not c) **FALSE** **TRUE**

PYTHON

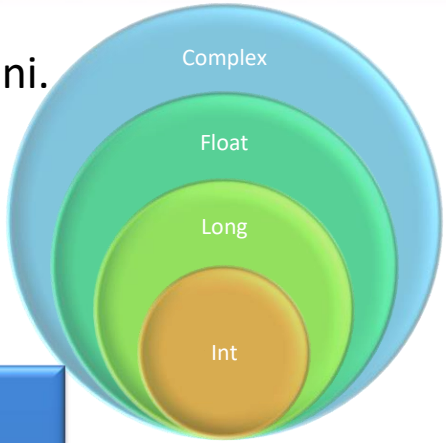
SET ISTRUZIONI : stringhe

Simbolo	Significato	Esempio
min	Restituisce il carattere minimo (ordine lessicografico)	str="Ciao" print(min(str)) **C**
max	Restituisce il carattere massimo (ordine lessicografico)	str="Ciao" print(max(str)) **o**
in	Restituisce TRUE se è presente una sottostringa in una stringa; FALSE altrimenti	str="Ciao" print('a' in str) **TRUE**
not in	Restituisce TRUE se NON è presente una sottostringa in una stringa; FALSE altrimenti	str="Ciao" print('k' not in str) **TRUE**

PYTHON

SET ISTRUZIONI : CONVERSIONI

La conversione tra tipi nativi viene eseguita attraverso specifiche funzioni.



Da stringa a valore numerico

intero=int(*stringa*)
reale= float(*stringa*)
complesso= complex(*stringa*)
intl= long(*stringa*)
intascii= ord(*carattere*)
#ORD restituisce il valore ASCII del carattere

Da tipo numerico a stringa

carattereascii=chr(*valoreascii*)
stringa=str(*valore*)

ESEMPI

```
str1="97"
print(int(str1))
97
str2="9.7"
print(float(str2))
9.7
str3="5"
print(ord(str3))
53 #NB: il carattere '0' ha valore SCII 48
str4="97"
print(eval(str4))
97
```

ESEMPI

```
val=49
print(chr(val))
'1'
val=67
print(chr(val))
'C'
val=3678543
print(str(val))
"3678543"
```

PYTHON

POLIMORFISMO DELLE FUNZIONI

In Python c'è un **polimorfismo dei tipi** ovvero il comportamento di una funzione o di un'operazione dipende esclusivamente dal tipo di oggetto su cui sono applicate

ESEMPIO

```
a=3
b=3*a
print (b)
9
a="ciao"
b=3*a
print(b)
"ciaociaociao"
```

PYTHON

POLIMORFISMO DELLE FUNZIONI

Il polimorfismo garantisce caratteristiche di concisione e di flessibilità del linguaggio. Essendo Python un linguaggio interpretato e con typing dinamico, il polimorfismo è intrinseco nel linguaggio.

ESEMPIO

```
def somma(a,b):
return a+b

print somma(3,4)
7
print somma('ciao',' mondo')
Ciao mondo
print somma([1,2],[3,3])
[1,2,3,3]
```


Fine