PROGRAMMAZIONE

Linguaggio Python
Strutture Dati

Dott. Franco Liberati
franco.liberati@unitus.it

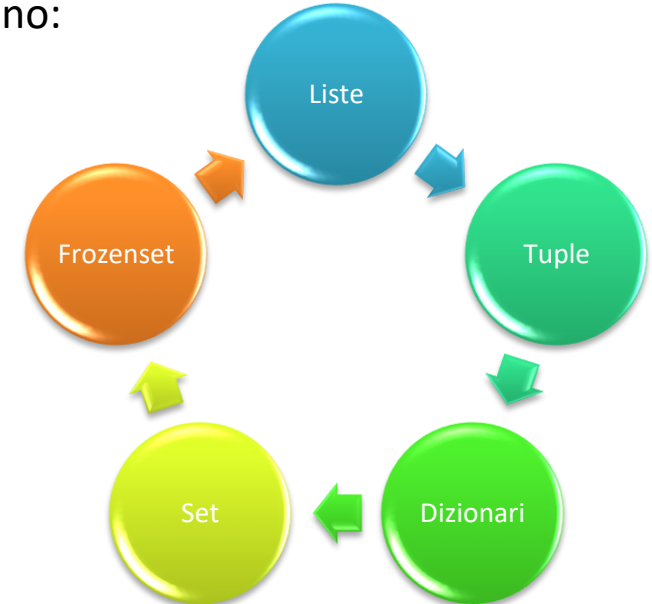
LINGUAGGIO PYTHON

Argomenti del corso

PYTHON

STRUTTURE DATI

- Python - oltre ai tipi di dato nativi del linguaggio come stringhe, interi, numeri in virgola mobile o complessi - fornisce delle **strutture dati native** del linguaggio (*built-in data structures*)
- Tali strutture si suddividono in sequenze ordinate di dati o collezioni di dati. Ma si possono anche classificare come strutture mutevoli e non-mutevoli
- Le *built-in data structures* in Python sono:



LISTE

PYTHON

LISTE

- La **list** è un tipo di dato strutturato nativo del linguaggio e definisce una sequenza ordinata di dati
- Gli elementi che costituiscono una lista possono essere dei valori relativi ad oggetti nativi del linguaggio (int, bool, float, string,...) o oggetti definiti dall'utente

- Una lista può contenere elementi di diverso tipo
- Una lista è un contenitori di dati modificabili

ESEMPIO

```
#Lista vuota
listavuota=[]

#Lista di 4 elementi (3 stringhe e un intero)
lista1=["Lista","di",4,"elementi"]
```

PYTHON

LISTE: prelievo ed aggiornamento elemento

- Gli elementi di una lista sono contenuti tra []
- Il primo elemento della lista ha indice zero
- Per accedere al singolo elemento della lista si ricorre all'operatore [] e si specifica la posizione con sintassi

  variabile=lista[*indice*]

- Per aggiornare un elemento si specifica l'indice con sinstassi

  lista[*indice*]=variabile/valore

ESEMPIO

```
#Lista di 4 elementi (3 stringhe e un intero)
lista=["Lista","di",4,"elementi"]

valore=lista[2]
print(valore)
4

lista[3]="elementi belli"
print(lista)
['Lista', 'di', 4, 'elementi belli']

lista[0]=1
print(lista)
[1, 'di', 4, 'elementi belli']
```

PYTHON

LISTE: prelievo ed aggiornamento elemento

- Le liste supportano l'operatore **slicing [**start:stop:step**]**

- L'operazione di slicing può essere applicata anche con un passo negativo (e, di solito, range decrescente)

ESEMPIO

```
lista=[0,1,2,3,4,5,6,7]

l1=lista[0:6]
print(l1)
[0, 1, 2, 3, 4, 5]

l2=lista[1:6:2]
print(l2)
[1, 3, 5]

l3=lista[1::2]
print(l3)
[1, 3, 5, 7]

l4=lista[::2]
print(l4)
[0, 2, 4, 6]
```

ESEMPIO

```
lista=[0,1,2,3,4,5,6,7]

l5=lista[6:0:-2]
print(l5)
[6, 4, 2]
```

PYTHON

LISTE: funzione RANGE

- La funzione range() è indicata per generare liste di numeri interi
- La sintassi è

range(start,stop,step**)**

e genera una lista di interi da start a stop-1 con passo step

- I numeri che eccedono lo stop non sono riportati

ESEMPIO

```
lista=range(3)
print(lista)
#formalmente
range(0, 3)
# In pratica ho 0,1,2
print(type(lista))
<class 'range'>
for n in lista:
  print(n)
0
1
2
```

PYTHON

LISTE: altre funzioni

Tipologia	Funzione	Significato	Esempio
Inserimento	*list.***append**(object)	inserimento di object in coda	fruits = ["apple", "banana", "cherry"] fruits.append("orange") print(fruits) **['apple', 'banana', 'cherry', 'orange']**
	*list.***insert**(index,object)	inserimento di object in posizione index	fruits = ['apple', 'banana', 'cherry'] fruits.insert(1, "orange") print(fruits) **['apple', 'orange', 'banana', 'cherry']**
	*list.***extend**(iterable)	concatenazione di liste	fruits = ['apple', 'banana', 'cherry'] cars = ['Ford', 'BMW', 'Volvo'] fruits.extend(cars) print(fruits) **['apple', 'banana', 'cherry', 'Ford', 'BMW', 'Volvo']**

PYTHON

LISTE: altre funzioni

Tipologia	Funzione	Significato	Esempio
Ricerca			
	list.**index**(value, [start, [stop]])	ricerca di value e ritorno della prima occorrenza	fruits = ['apple', 'banana', 'cherry'] x = fruits.index("cherry") print(x) **2**
	list.**count**(value)	calcolo delle occorrenze di value	fruits = ["apple", "banana", "cherry"] x = fruits.count("cherry") print(x) **1**

PYTHON

LISTE: altre funzioni

Tipologia	Funzione	Significato	Esempio
Eliminazione			
	*list.***remove**(value)	eliminazione della prima occorrenza di value	fruits = ['apple', 'banana', 'cherry'] fruits.remove("banana") **['apple', 'cherry']**
	*list.***pop**([index])	eliminazione elemento in posizione index	fruits = ['apple', 'banana', 'cherry'] fruits.pop(1) print(fruits) **['apple', 'cherry']**
Ordinamento			
	*list.***reverse**()	inversione dell'ordine degli elementi	fruits = ['apple', 'banana', 'cherry'] fruits.reverse() print(fruits) **['cherry', 'banana', 'apple']**
	*list.***sort**(cmp=None, key=None, reverse=False)	ordinamento	cars = ['Ford', 'BMW', 'Volvo'] cars.sort() print(cars) **['BMW', 'Ford', 'Volvo']**

PYTHON

LISTE: altre funzioni

Tipologia	Funzione	Significato	Esempio
Manipolazione			
	list+*list*	Concatenazione	fruits = ['apple', 'banana', 'cherry'] cars = ['BMW', 'Pegeout', 'Fiat'] unione=fruits+cars print(unione) **['apple', 'banana', 'cherry', 'BMW', 'Pegeout', 'Fiat']**
	*list*num*	Ripetizione	fruits = ['apple', 'banana', 'cherry'] rip=fruits*3 print(rip) **['apple', 'banana', 'cherry', 'apple', 'banana', 'cherry', 'apple', 'banana', 'cherry']**

TUPLE

PYTHON

TUPLE

- La ***tuple*** è una sequenza ordinata di dati
- La tuple è una lista i cui elementi sono NON mutabili (quindi non esistono le operazioni di eliminazione e di inserimento)
- La tuple è indicata per definire insiemi di valori costanti (*per alcune operazioni risultano più efficienti delle liste*)
- Una tuple si definisce esprimendo gli elementi tra **()**

ESEMPIO

```
#tupla vuota
tupla=()

#Tupla di 3 elementi
tupla=(1,2,3,4)
```

PYTHON

TUPLE: accesso

- L'accesso al singolo elemento di una tuple avviene attraverso l'operatore []
- Per estrarre elementi da una tuple è possibile usare lo slicing [start:end]

- NON è possibile modificare/aggiornare un elemento di una tuple

ESEMPIO

```
t1=(1,2,3,4,"ciao","mondo",[2,3])
t1[3]="inserisco una stringa a posto di 3"

Traceback (most recent call last):
File "<pyshell#26>", line 1, in <module>
t1[1]=3
TypeError: 'tuple' object does not support
item assignment
```

PYTHON

TUPLE: funzioni

- Sulle tuple sono definiti gli operatori **in** e **not in**

- È possibile convertire un dato tupla in un dato lista e viceversa rispettivamente attraverso le funzioni **list** e **tuple**

ESEMPIO

```
tupla=(3,4.5,7,8,'casa')
val=4.5;
print(val in tupla)
True

lista=[1,2,1,2]
tupla=tuple(lista)
print(tupla)
(1,2,1,2)
print(type(tupla))
<class 'tuple'>

tupla=(3,4.5,7,8,'casa')
lista=list(tupla)
print(lista)
[3, 4.5, 7, 8, 'casa']
print(type(lista))
<class 'list'>
```

DICTIONARY

PYTHON

DICTIONARY

- Un **dictionary** è un insieme non ordinato di oggetti i quali sono identificati univocamente da una chiave (***key***)
- I dati contenuti in un dizionario possono essere eterogenei
- Ogni elemento di un dictionary è quindi rappresentato da una coppia

  key – value

- In Python le chiavi possono essere solo oggetti immutabili: le tuple, per esempio, possono essere usate come chiavi di indicizzazione di un dictionary
- Per definire un dictionary si usano la sintassi

  indentificatore={}

ESEMPIO

```
# dizionario vuoto
d={ }

# dizionario con due elementi
d={1: "Hello", "due": "World"}

# dizionario NON valido
d={1: "Hello", "due": "World", 2:"Ciao", "due":
"Mondo"}
```

PYTHON

DICTIONARY: chiavi e valori

- Le chiavi in Python sono univoche e sono inoltre case-sensitive

- Le funzioni **key()** and **value()** ritornano rispettivamente le chiavi e i valori di un dizionario

ESEMPIO

```
d={"a":"Ciao",'b':"Mondo", "B":"Bello"}
d["a"]="CIAO A TUTTI";
print(d)
{'a': 'CIAO A TUTTI', 'b': 'Mondo', 'B': 'Bello'}

d["c"]
# Errore la chiave non è presente
Traceback (most recent call last):
File "<pyshell#73>", line 1, in <module>
d["c"]
KeyError: "c"

print(d.key())
["a", "b" , "B"]
print(d.value())
["CIAO A TUTTI", "Mondo", "Bello"]
```

PYTHON

DICTIONARY: aggiunta elementi

- Un dizionario può essere modificato aggiungendo elementi o modificando quelli esistenti
- I metodi più noti sono:

update(E, **F)

dizionario[nuovoindice]=valore

ESEMPIO

```
d={1:"Laurato",2:"Vlaovich",3:"Lukaku",4:"Immobile",5:"Leao" }

d.update({3:"Dybala"})
print(d)
{1: 'Laurato', 2: 'Vlaovich', 3: 'Dybala', 4: 'Immobile', 5: 'Leao'}

d[4]="Piola"
print(d)
{1: 'Laurato', 2: 'Vlaovich', 3: 'Dybala', 4: 'Piola', 5: 'Leao'}

d[2]="Platini"
print(d)
{1: 'Laurato', 2: 'Platini', 3: 'Dybala', 4: 'Piola', 5: 'Leao'}
```

PYTHON

DICTIONARY: cancellazione elementi

- È possibile cancellare un elemento o l'intero dizionario
- I metodi più noti sono:

diz.**pop(**k[,d]**)** #toglie un elemento con chiave k

diz.**popitem()** #estare l'ultimo elemento

diz.**clear()** #Cancella l'intero dizionario

ESEMPIO

```
d={1:"Mon",2:"Tue",3:"Wed",4:"Thu",5:"Fri",6:"Sat",7:"Sun"}
elem1=d.popitem()
print(elem1)
(7, 'Sun')
print(d)
{1: 'Mon', 2: 'Tue', 3: 'Wed', 4: 'Thu', 5: 'Fri', 6: 'Sat'}

elem2=d.pop(3)
print(elem2)
Wed
print(d)
{1: 'Mon', 2: 'Tue', 4: 'Thu', 5: 'Fri', 6: 'Sat'}

del d[4]
print(d)
{1: 'Mon', 2: 'Tue', 5: 'Fri', 6: 'Sat'}

d.clear()
print(d)
{}
```

PYTHON

DICTIONARY: altri metodi

- È possibile manipolare i dati dei dizionari usando delle altre funzioni

diz.**get**(k[,d]) #restituisce l'elemento indicato da k

diz.**has_key**(k) #se k è una chiave da TRUE altrimenti FALSE

diz.**items()**#resituisce gli elementi di un dizionario in una lista

ESEMPIO

```
d={1:"Mon",2:"Tue",3:"Wed",4:"Thu",5:"Fri",6:"Sat",7:"Sun"}

elem=d.get(7)
print(elem)
Sun

k_val=d.has_key(8)
print(k_val)
FALSE

lista1=d.items()
print(lista1)
[(1, 'Mon'), (2, 'Tue'), (3, 'Wed'), (4, 'Thu'), (5, 'Fri'), (6, 'Sat'), (7, 'Sun')]
```

PYTHON

DICTIONARY: costruzione matrice

- Un utilizzo interessante dei dizionari consiste nell'immagazzinamento di matrici sparse
- Nel formato coordinate si immagazzinano le coppie (i,j) corrispondenti ai valori a_{ij} non nulli

ESEMPIO

```
d={(0,0):1,(1,2):1,(2,0):2}

print(d)

#Formalmente
{(0, 0): 1, (1, 2): 1, (2, 0): 2}

#Logicamente
1       0       0
0       0       1
2       0       0
```

SET

PYTHON

SET

- Un **set** è un insieme (una collezione) mutabile non ordinata di oggetti
- Gli elementi, per definizione di insieme, sono univoci (non possono esserci elementi uguali)
- Per definire un set si ricorre alla sintassi

nomeinsieme=**set(**(value1,value2,..,valuen))

ESEMPIO

```
# insieme vuoto
empty=set(())
print(empty)
()

s=set(("ciao",1,"Mondo"))
print(s)
("ciao",1,"Mondo")
```

PYTHON

SET: modifica elementi

Tipologia	Funzione	Significato	Esempio
Inserimento	*set.***add**(object)	Aggiunge un elemento nell'insieme (se l'elemento già esiste è ignorato)	thisset = {"apple", "banana", "cherry"} thisset.add("orange") print(thisset) **{'cherry', 'banana', 'apple', 'orange'}** thisset.add("banana") print(thisset) **{'banana', 'cherry', 'apple', 'orange'}**
	*set.***update**(object)	Aggiunge un elemento/insieme ad un altro insieme (se un elemento già esiste è ignorato)	thisset = {"apple", "banana", "cherry"} tropical = {"pineapple", "mango", "papaya"} thisset.update(tropical) print(thisset) {'apple', 'mango', 'cherry', 'pineapple', 'banana', 'papaya'}

PYTHON

SET: modifica elementi

Tipologia	Funzione	Significato	Esempio
Eliminazione			
	*set.***discard**(x)	Rimuove un elemento (se l'elemento non esiste non da errore)	thisset = {"apple", "banana", "cherry"} thisset.discard("banana") print(thisset) **{"apple", "cherry"}**
	*set.***remove**(x)	Rimuove un elemento (se l'elemento non esiste da un errore)	thisset = {"apple", "banana", "cherry"} thisset.remove("banana") print(thisset) **{"apple", "cherry"}**
	*set.***pop**()	Rimuove un elemento a caso dall'insieme	thisset = {"apple", "banana", "cherry"} x = thisset.pop() print(x) print(thisset)
	*set.***clear**()	Cancella l'insieme	

PYTHON

SET: operazione tra insiemi

Tipologia	Funzione	Significato	Esempio
Operazioni			
	set.**union**(*set*)	Unione tra insiemi	x = {"apple", "banana", "cherry"} y = {"google", "microsoft", "apple"} z = x.union(y) print(z) **{'banana', 'microsoft', 'google', 'apple', 'cherry'}**
	set.**intersection**(*set*)	Intersezione tra insiemi	x = {"apple", "banana", "cherry"} y = {"google", "microsoft", "apple"} z = x.intersection(y) print(z) **{'apple'}**
	set.**difference**(*set*)	Differenza tra insiemi	x = {"apple", "banana", "cherry"} y = {"google", "microsoft", "apple"} z = x.difference(y) print(z)

PYTHON

SET: operazione tra insiemi

Tipologia	Funzione	Significato	Esempio
Operazioni			
	set1.**issubset**(*set2*)	Da True se l'insieme set1 è sottoinsieme di set2	x = {"a", "b", "c"} y = {"f", "e", "d", "c", "b", "a"} z = x.issubset(y) print(z) **True**
	set2.**issuperset**(*set1*)	Da True se l'insieme set2 è sottoinsieme di set1	x = {"f", "e", "d", "c", "b", "a"} y = {"a", "b", "c"} z = x.issuperset(y) print(z) **True**

SET FROZEN

PYTHON

SETFROZEN

- Un **setfrozen** è un insieme (una collezione) NON mutabile non ordinata di oggetti
- Gli elementi, per definizione di insieme, sono univoci (non possono esserci elementi uguali)
- Per definire un setfrozen si ricorre alla sintassi

nomeinsieme=**frozenset(**(value1,..,valuen))

I valori di un frozenset non possono essere modificati

ESEMPIO

```
mylist = ['apple', 'banana', 'cherry']
x = frozenset(mylist)
print(x)

frozenset({'apple', 'cherry', 'banana'})
```

```
mylist = ['apple', 'banana', 'cherry']
x = frozenset(mylist)
x[1] = "strawberry«
```

ERRORE

PYTHON

SET: operazione tra insiemi

Tipologia	Funzione	Significato	Esempio
Operazioni			
	set.**union**(*set*)	Unione tra insiemi	x = {"apple", "banana", "cherry"} y = {"google", "microsoft", "apple"} z = x.union(y) print(z) **{'banana', 'microsoft', 'google', 'apple', 'cherry'}**
	set.**intersection**(*set*)	Intersezione tra insiemi	x = {"apple", "banana", "cherry"} y = {"google", "microsoft", "apple"} z = x.intersection(y) print(z) **{'apple'}**
	set.**difference**(*set*)	Differenza tra insiemi	x = {"apple", "banana", "cherry"} y = {"google", "microsoft", "apple"} z = x.difference(y) print(z)

PYTHON

SET: operazione tra insiemi

Tipologia	Funzione	Significato	Esempio
Operazioni			
	set1.**issubset**(*set2*)	Da True se l'insieme set1 è sottoinsieme di set2	x = {"a", "b", "c"} y = {"f", "e", "d", "c", "b", "a"} z = x.issubset(y) print(z) **True**
	set2.**issuperset**(*set1*)	Da True se l'insieme set2 è sottoinsieme di set1	x = {"f", "e", "d", "c", "b", "a"} y = {"a", "b", "c"} z = x.issuperset(y) print(z) **True**

Cicli iterativi

PYTHON

Cicli FOR con oggetti iterabili

- Il ciclo for consente di iterare su oggetti iterabili come lista,tuple,stringhe, set, dizionari.

LISTE

```
lis1=[1,2,3,4,5]
for el in lis1:
        print el
```

```
1
2
3
4
5
```

STRINGHE

```
str1="Ciao"
for el in str1:
        print el
```

```
C
i
a
o
```

SET

```
set1=set([1,2,3,4])
for el in set1:
        print el
```

```
1
2
3
4
```

PYTHON

Cicli FOR con oggetti iterabili

DIZIONARI chiave

```
dic={1:'a',2:'b'}
for el in dic.keys():
        print el

1
2
```

DIZIONARI valori

```
dic={1:'a',2:'b'}
for el in dic.values():
        print el
a
b
```

DIZIONARI chiave-valori

```
dic={1:'a',2:'b'}
for k,v in dic.items():
        print k,v

1 a
2 b
```

DIZIONARI

```
dic={1:'a',2:'b'}
for el in dic:
        print el

1
2
```

DIZIONARI

```
dic={1:'a',2:'b'}
for el in (1,2,3):
        print dic.get(el)
a
b
None
```


Fine