Basi di Dati
Modulo Laboratorio

Lezione 9: Introduzione a Python per differenze con Java

DR. SARA MIGLIORINI

Linguaggio Python - Introduzione

- Python è un linguaggio di programmazione ad alto livello introdotto nel 1991.
- Caratteristiche strutturali essenziali: linguaggio multi paradigma, interpretato, con tipizzazione dinamica.
- Proprietà riconosciute:
  - sintassi più semplice di molti altri linguaggi,
  - disponibilità di strutture dati ad alto livello versatili,
  - possibilità di esprimere concetti/costrutti in meno righe rispetto a quanto possibile con altri linguaggi,
  - permette di essere più produttivi rispetto ad altri linguaggi nello sviluppo di codice.
- In questa lezione si introduce Python per differenze con il linguaggio Java.

Linguaggio Python – Comparazione con Java

Caratteristica	Python	Java
Tipo di programmazione	Supporta quasi tutti gli aspetti di programmazione orientata agli oggetti. Si possono scrivere programmi senza oggetti.	Supporta solo programmazione ad oggetti.
Tipo di esecuzione	Pensato per essere usato iterativamente tramite interprete (come SQL).	Programmi Java devono essere compilati esplicitamente (.class file). Class file vengono interpretati.

Linguaggio Python – Comparazione con Java

Caratteristica	Python	Java
Gestione tipi di dati	DINAMICA • Una variabile è definita assegnando un valore: pippo = 42 • Una variabile può cambiare tipo in un programma: v = 2 v = 'Due' ~~v = 'Due' + 2~~	STATICA • Una variabile deve essere dichiarate esplicitando il tipo prima dell'uso: int v = 2; • Una variabile NON può cambiare tipo in un programma: int v = 2; ~~v = "Due";~~

Linguaggio Python – Comparazione con Java

Caratteristica	Python	Java
Tipi di dati scalari	• int (unlimited precision) • bool • float (double precision) • complex	• byte (1 byte), short (2 byte), int (4 byte) • long (8 byte) • boolean • float/real • char (2 byte)
Strutture dati native	• list • tuple • range • str • set • dict	• java.util.List • ? • ? • String • java.util.Set • java.util.Map

Linguaggio Python – Comparazione con Java

Caratteristica	Python	Java
Delimitatore istruzioni	Carattere di fine riga termina un'istruzione. '\' per spezzare un'istruzione su più righe. Indentazione delimita un blocco (usare TAB o spazi, ma non mescolare!): i = 10 nomeFunzioneLungo (a, \ b, e, f) while i > 10: __i -= 1	Carattere ';' termina un'istruzione. Più istruzioni possono stare su una riga.

Linguaggio Python – Comparazione con Java

Caratteristica	Python	Java
Commenti	'#' inizia un commento di una riga.	'//' inizia un commento di una riga. '/*' per iniziare e '*/' per terminare commenti anche su più righe.

Linguaggio Python – Comparazione con Java

Caratteristica	Python	Java

Lista	Elenco di oggetti anche di tipi diversi.
	
	l = [1, 'uno', 2]
	l[0] # I elem.
	l[1:3] # dal II al III
	l[1] = 1
	l = list ()
		import java. util .*;
		
		List <Object> l = new ArrayList<>();
		l.add (1);
		l.add("uno");
		l.add(2);
		l.get(0);
		l.subList(1,3);
		l.set(1,1);

Linguaggio Python – Comparazione con Java

Caratteristica	Python	Java

Tupla	Lista di valori immutabile.
	
	t = (1, 'uno ', 2)
	t[0] # I elem .
	t[1:3] # dal II al III
	t = tuple ()
	t[1] = 2	

Linguaggio Python – Comparazione con Java

Caratteristica	Python	Java
Range	Lista di interi immutabile. range(1, 6) # (1 ,2 ,3 ,4 ,5) range(1, 8, 2) # (1 ,3 ,5 ,7) range(0, -6, -2) # (0 , -2 , -4)	

Linguaggio Python – Comparazione con Java

Caratteristica	Python	Java
Stringhe	Coppia di ‘ ’ o di " " delimitano stringhe. Coppia di """ """ delimita stringhe su più righe: 'Sono una stringa ' " Anch 'io" """ Io sono più lunga """	Coppia di ‘ ’ delimita un carattere. Coppia di " " delimita una stringa.

Linguaggio Python – Comparazione con Java

Caratteristica	Python	Java

Insieme	È una lista senza elementi ripetuti e senza un ordine fissato:
	
	s = {'Bob', 3, 4}
	2 in s # false
	3 not in s # false
		import java. util.*;
		
		Set<Object> s = new HashSet<>();
		l.add("Bob");
		s.add(3) ;
		s.add(4) ;
		s.contains(s); // false
		!s.contains (3); // false

Linguaggio Python – Comparazione con Java

Caratteristica	Python	Java
Dizionario	Array associativo. Chiave solitamente è un numero/stringa: d = {'name': 'Bob ', \ 3:4} d['name'] # è Bob d[3] # è 4 'name' in d #True	La struttura dati più simile è java.util.HashMap.

Linguaggio Python – Comparazione con Java

Caratteristica	Python	Java
Conversioni	Dato che è fortemente tipizzato, espressioni con tipi che non siano estensioni uno dell'altro non sono ammesse. Esistono molte funzioni predefinite per convertire da un tipo all'altro. Vedere http://www.tutorialspoint.com/python/python_variable_types.htm	

Linguaggio Python - Comparazione con Java

Caratteristica	Python	Java
Operatori logici	not/and/or	!/&&/||
Operatori di comparazione	>, <, >=, <=, ==, != si applicano anche a stringhe e altri tipi di oggetti: 'Tre' > "Due" # True [1, 2, 2] == [2, 2, 1] # False {1, 2, 3} == {3, 2, 1} # True {'a', 'b'} < {1, 2, 3} # Sottoinsieme\ False	>, <, >=, <=, ==, != si applicano solo ai tipi primitivi. ==, != per confrontare oggetti ma...

Linguaggio Python – Comparazione con Java

Caratteristica	Python	Java
Elemento vuoto	None Esempio: obj = None	null

Linguaggio Python – Comparazione con Java

Caratteristica	Python	Java

Cicli	for i in range (1, 10):
	__print(i) # i = 1, 2,... , 9
	
	for i in range (1, 10, 2):
	__print(i) # i=1, 3, 5, 7, 9
	
	for letter in 'Python':
	__print ('Letter : ' + letter)
	
	fruits = ['peach', 'apple', 'mango']
	for fruit in fruits :
	__print ('Fruit : '+ fruit)
		for(int i=1; i <10; i++)
		  System.out.println(i);
		
		for(int i=1; i <10; i +=2)
		  System.out.println(i);
		
		for (char c : " Java".toCharArray())
		  System.out.println(c);
		
		String[] fruits =
		{"peach","apple","mango"};
		for(String f : fruits)
		  System.out.println(f);

Linguaggio Python – Comparazione con Java

Caratteristica	Python	Java

Cicli While	while x > 0:
	    …
		while(x > 0){
		…
		}
		
		do {
		…
		} while(x > 0)

Linguaggio Python – Comparazione con Java

Caratteristica	Python	Java

Istruzione
if
	if x < 0:
	___...
	___...
	
	if x < 0:
	___...
	else :
	___...
	
	if x < 0:
	___...
	elif x > 0:
	___...
	else :
	___...
		if (x < 0) {
		...;
		...;
		}
		
		if (x < 0)
		...;
		else
		...;
		
		if (x < 0)
		...;
		else if (x > 0)
		...;
		else
		...;
		
		C'è anche lo switch

Linguaggio Python – Comparazione con Java

Caratteristica	Python	Java
Stampare su console	In Python 3.*: print(<var>/ cost, ...)	System.out.println(...); System.out.print(...); System.out.printf(...);

Leggere da console	In Python 3.*:
	i = input("Inserire numero: ")
	i = int(i)
	
	Il tipo del dato letto è sempre stringa!
		// Java 7
		Scanner keyb = new Scanner(System.in);
		System.out.println(“Enter an integer ");
		int i = keyb.nextInt();

Linguaggio Python – Comparazione con Java

Caratteristica	Python	Java

File I/O	f = open('workfile', 'r')
	# altri modi: r+, w, w+, ..
	for line in f:
	__print (line, end='')
	l = f.read() # legge tutto!
	l = f.readline() # legge una riga
		File d = new File ("workfile“)
		FileReader fr = new FilesReader(f);
		BufferedReader in = new
		BufferedReader(fr);
		String line = null;
		while((line = in.readLine ()) != null) {
		  System.out(line);
		}
		in.close ();
		fr. close ();

Linguaggio Python – Comparazione con Java

Caratteristica	Python	Java

Definizione di funzione	def nome(parametri):
	__corpo1
	__corpo2
	
	Esempio :
	def disc(a, b, c):
	    return b*b - 4*a*c
		Esempio :
		class Solver {
		  double disc(double a, double b, double c){
		      return b*b - 4*a*c;
		  }
		}

Linguaggio Python - Comparazione con Java

Caratteristica	Python	Java

Classi	Esempio:
	class Animal ():
	__""" A simple class """
	__def __init__(self , name):
	__ __self.name = name
	
	__def say(self):
	__ __print ("I am", self.name)
	
	class Dog(Animal):
	__""" A simple subclass """
	__def say(self):
	__ __print ("I am", self.name , \
	"and I can bark !")
	
	d = Dog('Pippo')
	d.say ()
		class Animal {
		private String name;
		public Animal(String name) {
		this.name = name ;
		}
		public say () {
		System.out.println("I am " + name);
		}}
		class Dog extends Animal {
		public Dog(String name){
		super(name);
		}
		public say() {
		System.out.println("I am " + name +
		" and I can bark !");
		}
		static public void main(String[] args) {
		Dog d = new Dog(" Pippo");
		d.say();
		}}

Linguaggio Python – Python 2 vs 3

- Python 3.13.3 è l'ultima versione disponibile. Non è compatibile con le versioni 2.x. È stata introdotta per correggere alcuni ambiguità del linguaggio e per razionalizzare alcuni costrutti.
- Python 2.7 è l'ultima versione della serie 2.

Cosa	Python 2	Python 3
print	istruzione	funzione
divisione intera	3/2=1	3/2=1.5
String	lista di caratteri ASCII	lista di caratteri UTF8!
certi valori di funzioni hanno cambiato tipo	dicts.keys(), dict.values() ritornano liste.	ritornano "viste" sullo stesso oggetto.
	map() e filter() ritornano liste	map() e filter() ritornano iteratori.
Operatori di confronto	(<, <=, >=, >) con tipi eterogenei ritorna false	(<, <=, >=, >) con tipi eterogenei solleva un'eccezione.
Sintassi	Alcune keyword sono state cambiate o eliminate!	
Molti altri cambiamenti più specifici!		

Linguaggio Python - Documentazione

- Riferimento principale: https://docs.python.org/3/
- Libro consigliato per chi vuole approfondire: Mark Lutz. Learning Python. https://learning-python.com/about-lp5e.html
- Sito di tutorial specifici su caratteristiche del linguaggio: https://pythonspot.com/en/all-tutorials/

Linguaggio Python – Ambiente di lavoro

- Un'installazione standard di Python 3 (https://www.python.org/) rende disponibile un'interprete, python, che è un'interprete da console molto simile a psql.
- In molte distr. di Linux e in Mac OS X, Python 2 è già presente. Talvolta anche Python 3. Installare Python 3 se necessario!
- Quando Python 3 è installato, è presente l'interprete python3:
  - python potrebbe essere collegato a Python 2. Usare python3 per essere certi di usare Python 3.

Linguaggio Python - Ambiente di lavoro

Python in laboratorio Delta

```
$ python3
Python 3.6.4 ( default , Mar 9 2018 , 23:15:03)
[GCC 4.2.1 Compatible Apple LLVM 9.0.0 (clang -900.0.39.2)]
Type "help", "copyright", "credits" or "license" for more
information.
>>> print ("Hello world!")
Hello world!
>>> exit ()
$
```

Linguaggio Python - Ambiente di lavoro

- Analogamente a quanto fatto per psql, dopo i primi esperimenti, conviene scrivere il proprio programma Python in un file ed eseguire il file.
- Solitamente i programmi python hanno suffisso .py.
- python3 <filePython> interpreta il file filePython come programma Python 3 e termina.
- python3 -i <filePython> interpreta il file filePython come programma Python 3 e rimane aperto.
- Con Python 3, un file di programma deve essere scritto usando UTF8.

Python – Esempio Fibonacci

File fibonacci.py

```
# Serie di Fibonacci
a, b = 0, 1 # assegnamento in sequenza
n = int( input ("Fino a quale intero calcolare la serie ?"))
while b < n:
__print (b, end='')
__a, b = b, a+b
print()
```

Esecuzione di fibonacci.py

```
$ python3 fibonacci.py
Fino a quale intero calcolare la serie ? 100
1 1 2 3 5 8 13 21 34 55 89
$
```

Es. 1 – Scrittura di una tabella in un file 1/7

File scritturaDict.py:

```
"""
Si vuole salvare in un file una tabella delle spese definita come
table spese {
  data DATE ,
  voce VARCHAR ,
  importo Decimal
}
In questo modulo si rappresenta la tabella come
una lista di record ( dict ) e si usa il formato CSV per salvare .
"""
# ##################################################################
# IMPORT
import csv
import re
```

Es. 1 – Scrittura di una tabella in un file 2/7

File scritturaDict.py:

```
# CLASSI o funzioni di servizio
pattern = re.compile ("^\d {2 ,2}/\ d {2 ,2}/\ d{4 ,4}$")

def creaDict( data , voce , importo ):
""" Ritorna un dict formato con gli argomenti .
Formato: {'data': <data>, 'voce': <voce>, 'importo': < importo >}
Fa il check sul formato data.
"""
if not isinstance (data , str) or not pattern.match ( data ):
    print( "Data non è nel formato dd/mm/ aaaa" )
    exit()
return { 'data': data , 'voce': voce, 'importo': float( importo ) }
```

Più avanti si avrà evidenza che lasciare «importo» a essere un tipo float è un errore anche quando le cose vanno bene! Inoltre, data è solo approssimata ad essere un tipo «Data»!

Es. 1 – Scrittura di una tabella in un file 3/7

File scritturaDict.py:

```
# Creo la tabella e aggiungo dati
tabella = list ()
tabella.append( creaDict ( "24/02/2016", "Stipendio", 0.1) )
tabella.append( creaDict ( "24/02/2016", 'Stipendio "Bis"', 0.1) )
tabella.append( creaDict ( "24/02/2016", 'Stipendio "Tris"', 0.1) )
tabella.append( creaDict ( "27/02/2016", "Affitto", -0.3))
# Stampo la tabella da memoria
print( '=' * 50 )
print( "| {:10 s} | {: <20} | {: >10s} |". format( "Data", "Voce", "Importo") )
print( '-' * 50 )
for riga in tabella :
    print( "| {:10 s} | {: <20} | {: >10.2 f} |". format ( riga ['data'], riga ['voce'], riga ['importo']))
    print ('=' * 50)
# Calcolo il totale degli importi
tot = 0.0
for riga in tabella:
    tot += riga ['importo']
print( "La somma è {:.20 f}". format (tot) )
```

Es. 1 – Scrittura di una tabella in un file 4/7

File scritturaDict.py:

```
# Salvo la tabella in un file in formato CSV
nomeFile = 'tabellaSpesa.csv'
with open( nomeFile, mode = 'w', encoding = 'utf -8 ') as csvFile:
    nomiCampi = ['data', 'voce', 'importo']
    writer = csv.DictWriter( csvFile , fieldnames = nomiCampi )
    writer.writeheader ()
    for riga in tabella :
        writer.writerow( riga )

# Leggo dal file la tabella e la pongo in una nuova variabile
tab1 = list()
with open( nomeFile , mode = 'r', encoding = 'utf -8 ') as csvFile :
    reader = csv.DictReader( csvFile )
    for row in reader :
        tab1.append ( creaDict (row['data'], row['voce'], row['importo']))
```

Es. 1 – Scrittura di una tabella in un file 5/7

File scritturaDict.py:

```
# Calcolo il totale sulla nuova tabella
tot1 = 0
for riga in tab1 :
    tot1 += riga ['importo']
if tot == tot1:
    print( "I due totali sono uguali !« )
else :
    print( " Ops ... la tabella letta non ha gli stessi dati !" )
if tot == 0:
    print( " Eureka !" )
else :
    print( " Ops ... il totale non è corretto perchè non è 0!" )
```

Es. – Scrittura di una tabella in un file 6/7

File tabellaSpesa.csv

```
data ,voce , importo
24/02/2016, Stipendio, 0.1
24/02/2016, "Stipendio ""Bis""", 0.1
24/02/2016, "Stipendio ""Tris""", 0.1
27/02/2016, Affitto , -0.3
```

Es. 1 – Scrittura di una tabella in un file 7/ 7

Output di una esecuzione

```
================================================
| Data       | Voce                 | Importo  |
------------------------------------------------
| 24/02/2016 | Stipendio            | 0.10     |
| 24/02/2016 | Stipendio "Bis"      | 0.10     |
| 24/02/2016 | Stipendio "Tris"     | 0.10     |
| 27/02/2016 | Affitto              | -0.30    |
================================================
```

La somma è 0.00000000000000005551
I due totali sono uguali !
Ops ... il totale non è corretto perché non è 0!

Es. 2 – Scrittura di una tabella in un file 1/12

- Il file scritturaClasse.py rappresenta un'evoluzione del file 'scritturaDict.py' in cui si risolve l'errore sugli importi e si dimostra un uso più avanzato di JSON:

File scritturaClasse.py:

```
"""
Si vuole salvare in un file una tabella delle spese definita come
table spese {
  data DATE ,
  voce VARCHAR ,
  importo Decimal
}
In questo modulo si rappresenta la tabella come un oggetto di una classe e si usa JSON per
salvare .
Inoltre gli importi sono rappresentati come Decimal .
"""
```

Es. 2 – Scrittura di una tabella in un file 2/12

File scritturaClasse.py:

```
# IMPORT
import datetime
import decimal
import json
import re
# ##################################################################
# CLASSI o funzioni di servizio
pattern = re.compile( "^\d {2 ,2}/\ d {2 ,2}/\ d{4 ,4}$" )
def creaDict( data, voce, importo ):
"""
Ritorna un dict formato con gli argomenti .
Formato : {' data ':<data >,' voce ':<voce >, importo :< importo > }.
Fa il check sul formato data e su importo che deve essere Decimal .
"""
```

Es. 2 – Scrittura di una tabella in un file 3/12

File scritturaClasse.py:

```
__if not isinstance( data, str ) or not pattern.match( data ):
__ __print( "Data non è nel formato dd/mm/aaaa" )
__ __exit()
__if not isinstance( importo , decimal.Decimal ) and not isinstance ( importo, str ):
__ __print ( "Importo deve essere un Decimal o una stringa che rappresenta un importo." )
__ __exit()
__return{ 'data': data, 'voce': voce, 'importo': decimal.Decimal ( importo ) }
```

Ora 'importo' è forzato ad essere di tipo Decimal, che ha proprietà simili Decimal di SQL

Es. 2 – Scrittura di una tabella in un file 4/12

File scritturaClasse.py:

```
class Spese :
    """
    L'attributo 'tabella' reppresenta le spese organizzate come dict( primaryKey, riga ).
    La primaryKey è data dalla coppia ( data, voce ).
    La riga è un dict creato dal metodo creaDict().
    L'attributo 'ultimaModifica' rappresenta il timestamp dell 'ultima modifica .
    """
    def makeKey (data , voce ):
        return data + "_%_" + voce

    def __init__ (self, inputTab = {} , istante = datetime.datetime.now()):
        self.tabella = dict ( inputTab )
        self.ultimaModifica = istante
```

Es. 2 – Scrittura di una tabella in un file 5/12

File scritturaClasse.py:

```
def add( self, data, voce, importo ):
    self.tabella[Spese.makeKey( data , voce )] = creaDict( data , voce , importo )
    self.ultimaModifica = datetime.datetime.now()
    return importo

def remove( self, data, voce ):
    del self.tabella[Spese.makeKey( data , voce )]
    self.ultimaModifica = datetime.datetime.now()

def get( self , data , voce ):
    return self.tabella[self.makeKey( data , voce )]

def items ( self ):
    return self.tabella.items()
```

Es. 2 – Scrittura di una tabella in un file 6/12

File scritturaClasse.py:

```
# ##################################################################
# Creo la tabella e aggiungo dati
tab = Spese ()
tab.add( "24/02/2016 ", " Stipendio ", "0.1")
tab.add( "24/02/2016 ", 'Stipendio "Bis"', "0.1" )
tab.add( "24/02/2016 ", 'Stipendio "Tris"', "0.1" )
tab.add( "27/02/2016 ", " Affitto ", " -0.3" )
# Stampo la tabella da memoria
print( '=' * 50 )
print( "| {:10 s} | {: <20} | {: >10s} |".format( "Data", "Voce", "Importo" ))
print( '-' * 50 )
for riga in tab.tabella.values():
    print( "| {:10 s} | {: <20} | {: >10.2 f} |".format( riga['data'], riga['voce'], riga['importo']))
    print( '=' * 50 )
```

Es. 2 – Scrittura di una tabella in un file 7/12

File scritturaClasse.py:

```
# Calcolo il totale degli importi
tot = decimal.Decimal(0)
for riga in tab.tabella.values():
    tot += riga['importo']
print ("La somma è {:.20 f}". format (tot))
```

Es. 2 – Scrittura di una tabella in un file 8/12

File scritturaClasse.py:

```
# Salvataggio in un file
class MyEncoder( json.JSONEncoder ):
    """ Codifica gli oggetti di tipo Decimal , Date o Spese ."""
    def default( self, o ):
        if isinstance( o, decimal.Decimal ):
            return float(o)
        if isinstance( o, datetime.datetime ):
            return o.isoformat()
        if isinstance( o, Spese ):
            return{ "tabella": o.tabella , "ultimaModifica": o.ultimaModifica }
        return json.JSONEncoder.default( self, o )

# Salvo la tabella in un file
nomeFile = 'database.json '
with open( nomeFile, mode ='w', encoding ='utf -8 ') as file :
    json.dump( tab, file, cls=MyEncoder, indent =4 )
```

Es. 2 – Scrittura di una tabella in un file 9/12

File scritturaClasse.py:

```
def myDecoder ( jsonObj ):
""" Decodifica oggetti di tipo Spesa ."""
    if 'tabella' in jsonObj :
        tab = jsonObj['tabella']
        istante = datetime.datetime.strptime( jsonObj['ultimaModifica'], "%Y -%m -% dT%H:%M:%S.%f" )
        return Spese (tab , istante )
    else :
        return jsonObj

# Leggo dal file la tabella e la pongo in una nuova variabile
with open( nomeFile , mode ='r', encoding ='utf -8' ) as file :
    tab1 = json.load( file , object_hook = myDecoder, parse_float = decimal.Decimal )
```

Es. 2 – Scrittura di una tabella in un file 10/12

File scritturaClasse.py:

```
# Calcolo il totale sulla nuova tabella
tot1 = decimal.Decimal(0)
for riga in tab1.tabella.values():
    tot1 += riga ['importo']
if tot == tot1 :
    print( "I due totali sono uguali !" )
if tot == 0:
    print( " Eureka !!" )
else :
    print( "Ops ... il totale non è corretto !" )
```

Es. 2 – Scrittura di una tabella in un file 11/12

File database.json (compattato)


```
{
   "tabella": {
      "24/02/2016 _% _StipendioTris":
         { "importo": 0.1, "data": "24/02/2016", "voce": "S... Tris« },
      "27/02/2016 _% _Affitto":
         { "importo": -0.3, "data": "27/02/2016", "voce": "Affitto" },
      "24/02/2016 _% _StipendioBis":
         { "importo": 0.1, "data": "24/02/2016", "voce": "Sti ... Bis"},
      "24/02/2016 _% _Stipendio":
         { "importo": 0.1, "data": "24/02/2016", "voce": " Stipendio" }
   },
   "ultimaModifica": "2016 -04 -15 T12 :06:50.899597"
}
```

Es. 2 – Scrittura di una tabella in un file 12/12

Output

```
================================================
| Data       | Voce              | Importo |
------------------------------------------------
| 27/02/2016 | Affitto           |   -0.30 |
| 24/02/2016 | Stipendio "Tris"  |    0.10 |
| 24/02/2016 | Stipendio         |    0.10 |
| 24/02/2016 | Stipendio "Bis"   |    0.10 |
================================================
La somma è 0.00000000000000000000
I due totali sono uguali !
Eureka !
```