UNIVERSITÀ PARTHENOPE
DiST
Dipartimento di Scienze e Tecnologie

Intelligent Signal Processing

Brief Introduction to Python

Angelo Ciaramella

Introduzione

- Linguaggio di programmazione moderno
  - sviluppo di siti o applicazioni Web e desktop
  - realizzazione di interfacce grafiche
  - amministrazione di sistema
  - calcolo scientifico e numerico
  - database giiochi
  - grafica 3D
  - …
- Linguaggio ABC
  - Primi anni 80
  - National Research Institute for Mathematics and Computer Science (CWI) di Amsterdam
  - tra i ricercatori **Guido Van Rossum**

Introduzione

- Linguaggio Python
  - Fine anni 80
  - Guido Van Rossum – miglioramento di ABC
  - Libro: *Programming Python*, 1996
  - Fan di «*Monty Python's Flying Circus*»
  - *2000 Python 2.0*
  - *2008 Python 3.0*

I punti di forza

- Free
  - completamente gratuito ed è possibile usarlo e distribuirlo senza restrizioni di copyright
- Multi-paradigma
  - Programmazione procedurale
  - Programmazione ad oggetti
  - Diversi elementi della programmazione funzionale
- Portabile
  - sviluppato in ANSI C
  - diverse piattaforme come: Unix, Linux, Windows, DOS, Macintosh, Sistemi Real Time, OS/2, cellulari Android e iOS

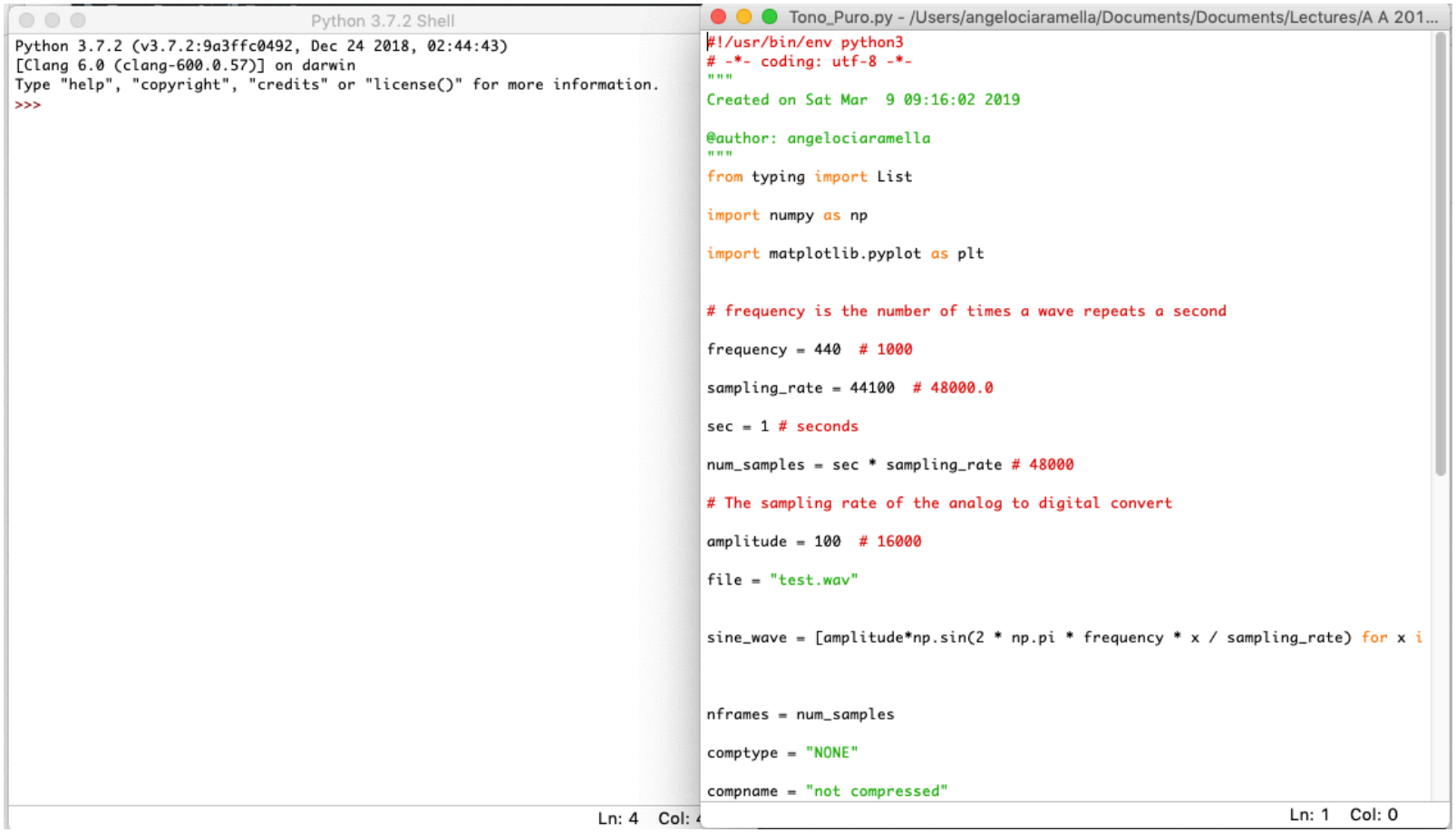
I punti di forza

- Interpetrato
  - lo stesso codice può essere eseguito su qualsiasi piattaforma purché abbia l'interprete Python installato
- Facile da usare
  - linguaggio di alto livello semplice e potente
- Ricco di librerie
  - standard library è una collezione di oltre 200 moduli
- Performante
  - i programmi vengono automaticamente compilati in *bytecode* prima di essere eseguiti

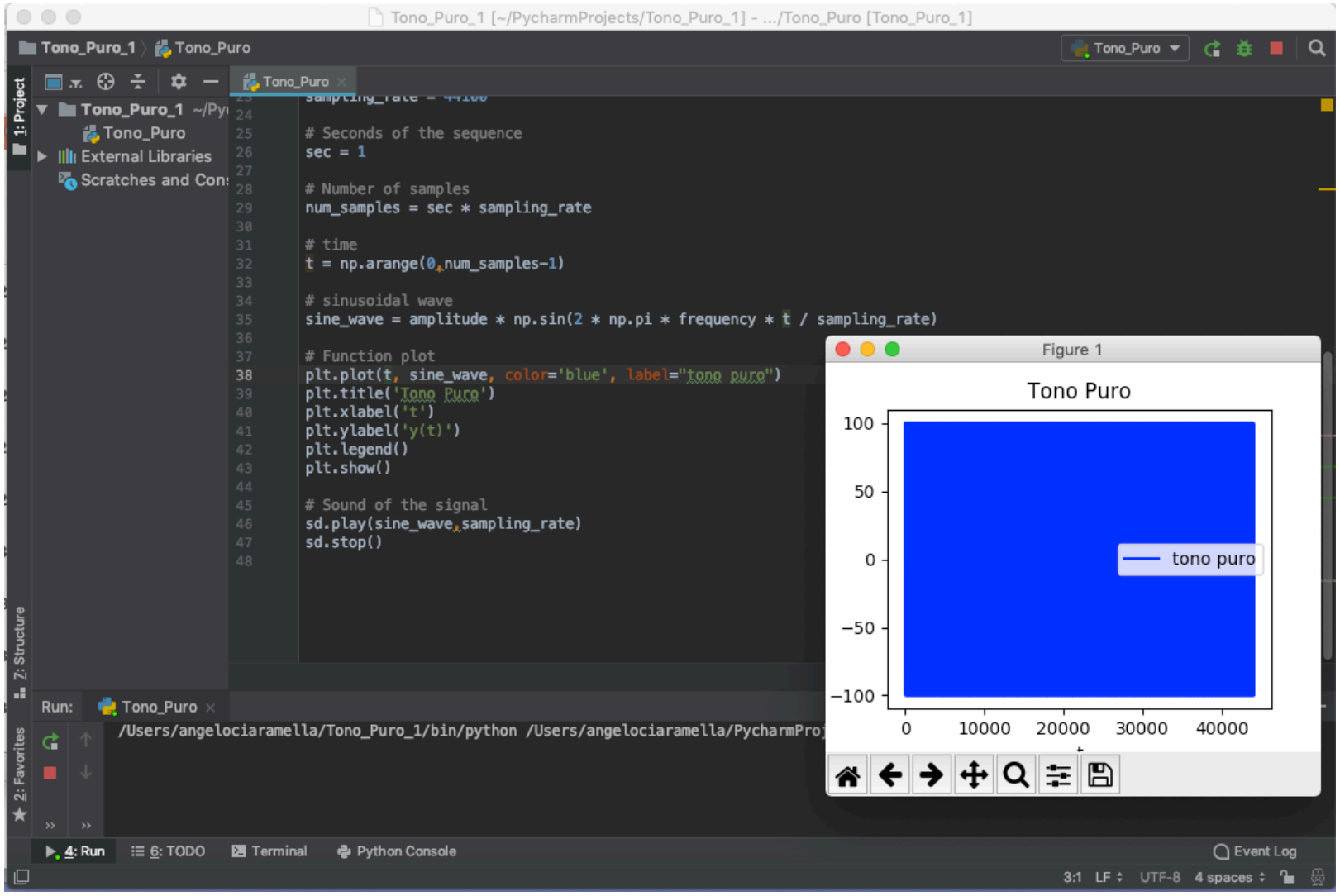
I punti di forza

- Gestisce automaticamente la memoria
  - *garbage collection*
- Integrabile con altri linguaggi
  - interprete classico scritto in C
  - .NET, Java

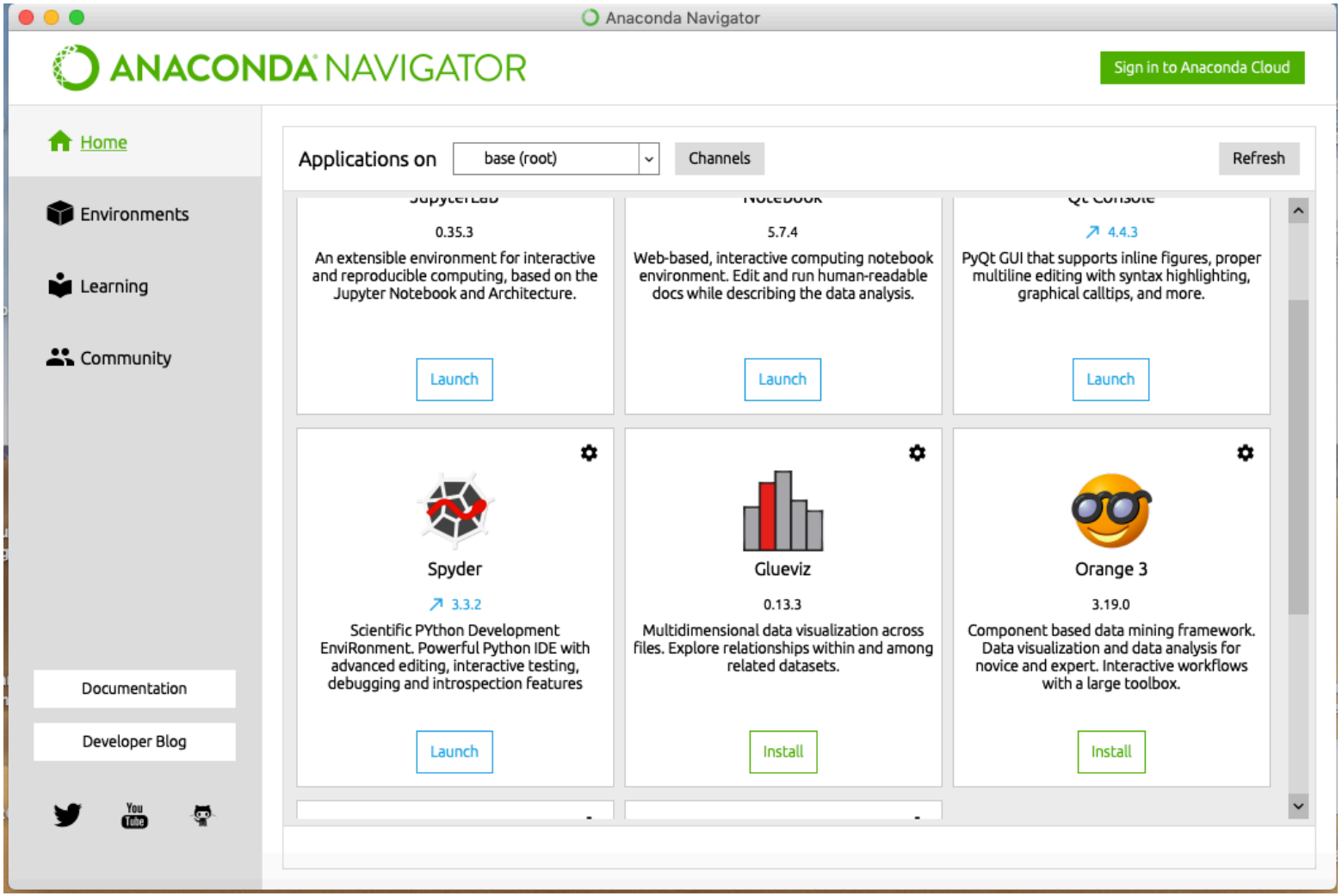
Python IDE: IDLE



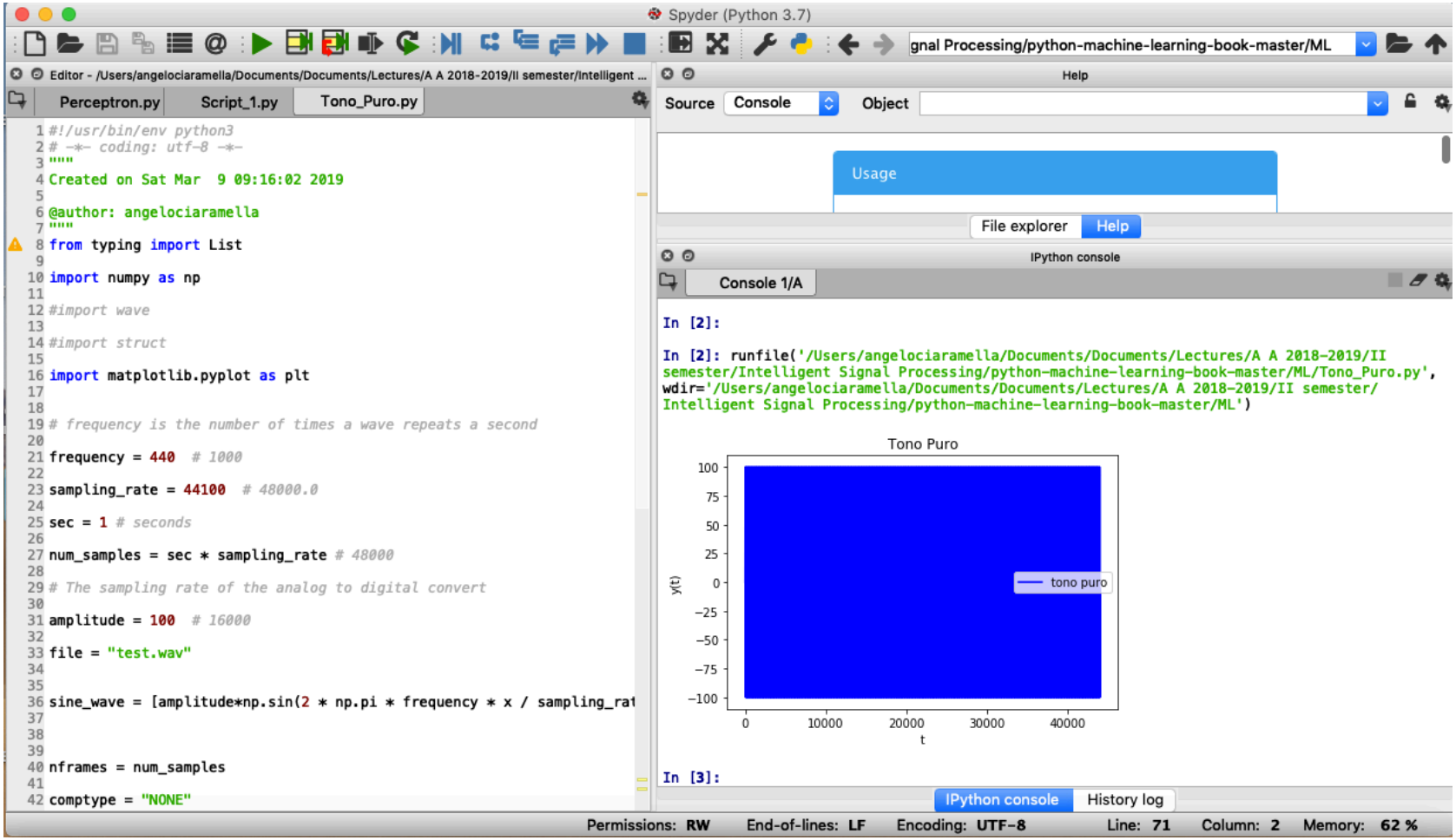
Python IDE: PyCharm



Python IDE: Anaconda



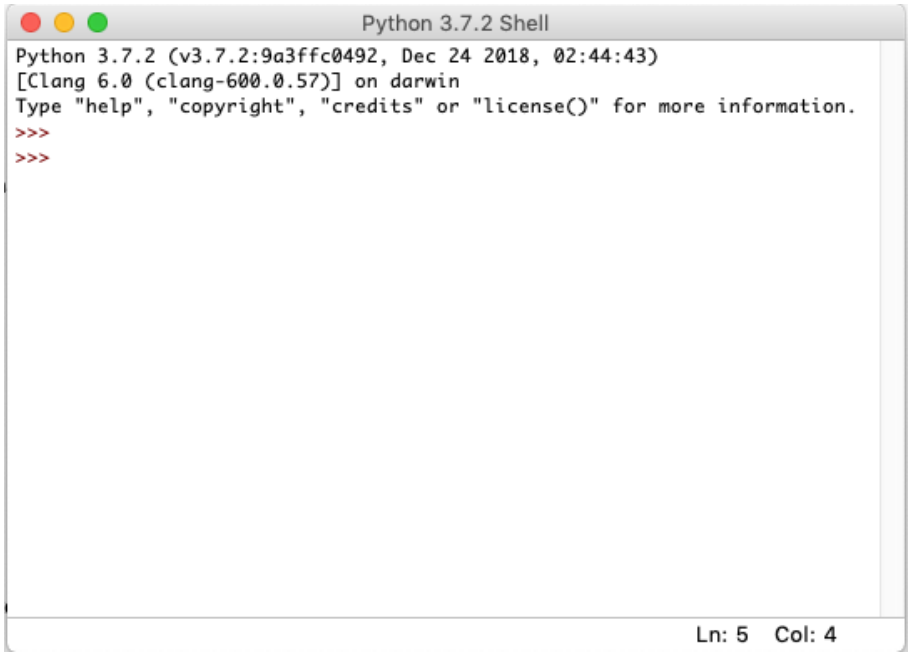
Python IDE: Spider



Interprete di Python

- Interprete interattivo
  - leggere e valutare man mano le espressioni inserite dall'utente
  - eseguire script contenenti sequenze di istruzioni Python

(script.py)



Interprete interattivo di python

Linea di comando

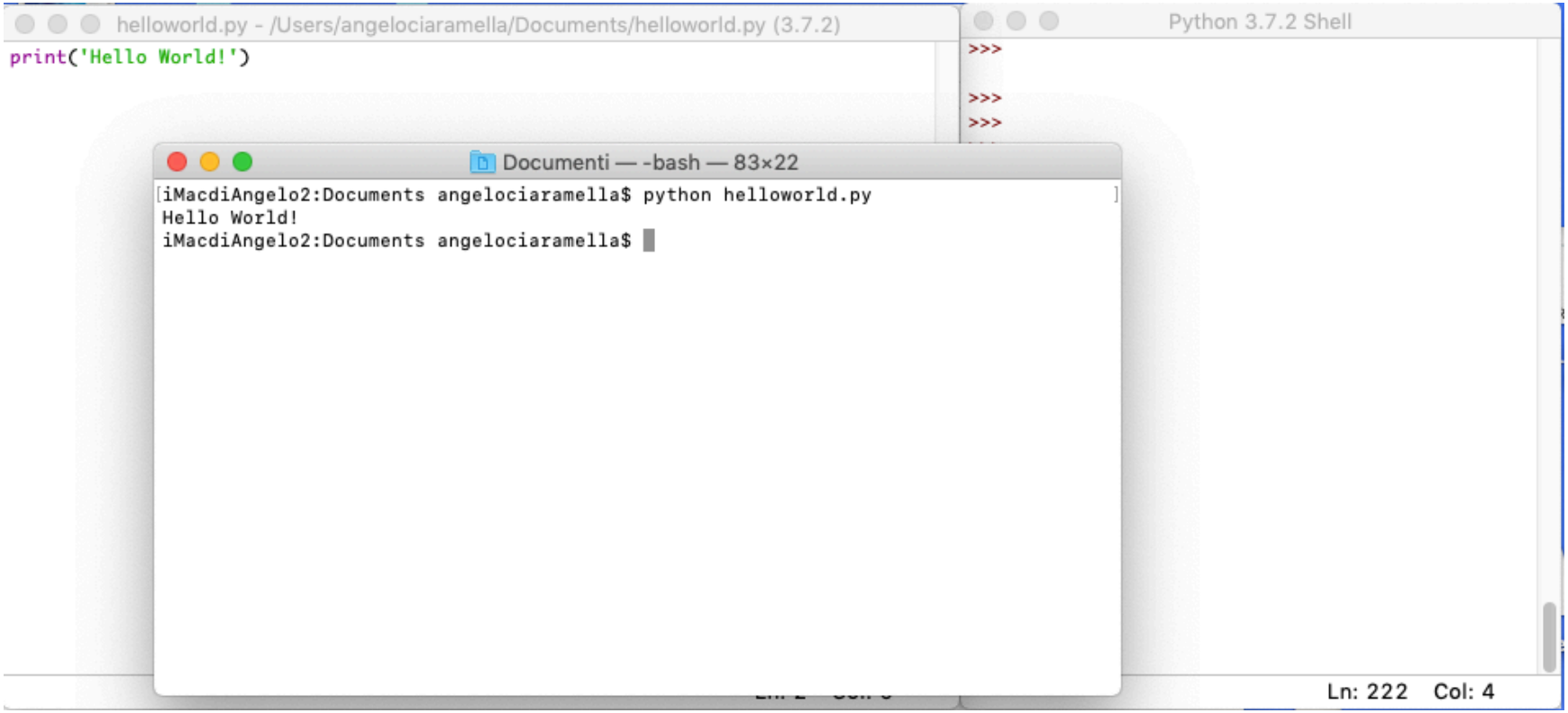
```
>>> 4*5
20
>>> a=4
>>> b=5
>>> 3 * (a + b) + 3 * (a - b)
24
>>>
```

Esempio di esecuzione da linea di comando

```
>>> print('Hello World!')
Hello World!
>>>
```

Esempio di esecuzione da linea di comando

Esecuzione da linea di comando



Esecuzione di un file .py

print

- input
  - numero variabile di argomenti (anche di tipi diversi) e li converte in stringhe

- output
  - mostra gli argomenti in output separati da uno spazio e seguiti da un carattere di ritorno a capo (\n)

```
>>> a = 10
>>> b = 3
>>> print(a, b, a - b)
10 3 7
>>> x = 'Ciao'
>>> print(c, '!!!')
Ciao!!!
```

Esempio di utilizzo della funzione print

input

- singolo argomento opzionale
  - stringa che viene mostrata a video prima di leggere il valore digitato

```
>>> nome = input('Inserisci il tuo nome: ‘)
Inserisci il tuo nome: Mario
>>> nome
‘Mario’
```

Esempio di utilizzo della funzione input

input

- valori numerici
  - convertire la stringa restituita da input usando funzioni come `int` o `float`

```
>>> raggio = input('Inserisci il raggio: ')
Inserisci il raggio: 3.1
>>> r = float(raggio)
>>> raggio, r
('3.1', 3.1)
>>>
```

Esempio di utilizzo della funzione input

Esercizio: calcolare e visualizzare l'area e il perimetro di un cerchio

Scrittura del codice

- Indentazione
  - a differenza di altri linguaggi Python usa l'indentazione

```
print("eseguito sempre all'inizio")
if condizione:
        print('eseguito in mezzo solo se la condizione è vera')
        print('eseguito in mezzo solo se la condizione è vera')
        print('eseguito in mezzo solo se la condizione è vera')
print('eseguito sempre alla fine')
```

Esempio di indentazione

Tipi di dati

Tipo di dato	Nome	Descrizione	Esempi
Intero	int	Intero di dimensione arbitraria	-42, 0, 1200, 999999999999999
Reale	float	Numero a virgola mobile	3.14, 1.23e-10, 4.0E210
Booleano	bool	Per valori veri o falsi	True, False
Complesso	complex	Numeri complessi con parte reale e immaginaria	3+4j, 5.0+4.1j, 3j
Stringhe	str	Usata per rappresentare testo	'', 'stefano', "l'acqua"
Bytes	bytes	Usata per rappresentare bytes	b'', b'\x00\x01\x02', b'Python'
Liste	list	Una sequenza mutabile di oggetti	[], [1, 2, 3], ['Hello', 'World']
Tuple	tuple	Una sequenza immutabile di oggetti	(), (1, 2, 3), ('Python', 3)
Insiemi	set/frozenset	Un'insieme di oggetti unici	{1, 2, 3}, {'World', 'Hello'}
Dizionari	dict	Una struttura che associa chiavi a valori	{}, {'nome': 'Ezio', 'cognome': 'Melotti'}

Variabili

- Oggetti
  - numero, stringa, lista, …
- Variabili
  - non hanno tipo
  - etichette che si riferiscono ad un determinato oggetto

```
x = 10
x = "Python"
x = [1, 2, 3]
```

Esempio di assegnazione di variabili

Variabili

```
>>> a, b, c = 2, 3, 5 # assegnamento multiplo
>>> a * b + c 11
>>> a
2
>>> b
3
>>> c
5
```

Esempio di assegnamento multiplo

Tipi numerici

```
>>> 6 + 4*3
18
>>> 9 / 2 # restituisce un float
4.5
>>> 2j * 2j # moltiplicazione tra numeri complessi
(-4+0j)
>>> int(10.6) # conversione esplicita
10
>>> float(20) # conversione esplicita
20.0
>>> 0b11111111 # intero in notazione binaria (prefisso 0b)
255
>>> 0o377 # intero in notazione ottale (prefisso 0o)
255
>>> 0xFF # intero in notazione esadecimale (prefisso 0x)
255
```


Esempio di assegnamento multiplo

Operatori aritmetici

Operatore	Descrizione	Esempi
+	addizione	10 + 12 → 22
–	sottrazione	5 - 1 → 4
*	moltiplicazione	10 * 12 → 120
/	divisione	9 / 4 → 2.25
//	divisione intera	9 // 4 → 2
%	modulo (resto della divisione)	9 % 4 → 1

Operatori aritmetici

Operatori di confronto

Operatore	Descrizione	Esempi
==	uguale a	8 == 8 → True 3 == 5 → False
!=	diverso da	3 != 5 → True 8 != 8 → False
<	minore di	3 < 5 → True 5 < 3 → False
<=	minore o uguale a	3 <= 5 → True 8 <= 8 → True
>	maggiore di	5 > 3 → True 3 > 5 → False
>=	maggiore o uguale a	5 >= 3 → True 8 >= 8 → True

Operatori di confronto

Operatori booleani

Operatore	Descrizione
and	Ritorna True se entrambi gli operandi sono *veri*, altrimenti False
or	Ritorna True se almeno uno degli operandi è *vero*, altrimenti False
not	Ritorna False se l'operando è *vero*, True se l'operando è *falso*

Operatori booleani

Operatori binari

Operatore	Descrizione
x << n	esegue uno *shift* a sinistra di n posizioni dei bit di x
x >> n	esegue uno *shift* a destra di n posizioni dei bit di x
x & y	esegue un *and* tra i bit di x e di y
x | y	esegue un *or* tra i bit di x e di y
x ^ y	esegue un *or esclusivo* tra i bit di x e di y
~x	inverte i bit di x

Operatori binari

Stringhe in Python

- Stringhe
  - è possibile racchiudere il suo valore indifferentemente tra apici (carattere ') o doppi apici (carattere ")

```
>>> s = 'Python'
>>> s[0] # elemento in posizione 0 (il primo)
'P'
>>> s[5] # elemento in posizione 5 (il sesto)
'n'
>>> s[-1] # elemento in posizione -1 (l'ultimo)
'n'
>>> s[-4] # elemento in posizione -4 (il quartultimo)
't'
```

Indexing (I numeri negative partono dall'ultimo elemento di indice -1)

Stringhe in Python

```
>>> s = 'Python'
>>> s[0:2] # sottostringa da 0 (incluso) a 2 (escluso)
'Py'
>>> s[:2] # dall'inizio all'elemento con indice 2 (escluso)
'Py'
>>> s[3:5] # elementi con indice 3 (incluso) a 5 (escluso)
'ho'
>>> s[4:] # elementi con indice 4 (incluso) alla fine
'on'
>>> s[-2:] # elementi con indice -2 (incluso) alla fine
'on'
```

Esempi di slicing

Stringhe in Python

- operatori
  - in – se un elemento fa parte di una sequenza
  - not in – se un elemento non fa parte di una sequenza

```
>>> s = 'Python'
>>> 'P' in s # 'P' è contenuto nella stringa s
True
>>> 'x' in s # il carattere 'x' non è in s
False
>>> 'x' not in s # "not in" esegue l'operazione inversa
True
>>> 'Py' in s # sottostringa 'Py' è nella stringa s
True
>>> 'py' in s # ?
```

Esempi di contenimento

Operazioni con le stringhe

```
>>> 'Py' + 'thon'
'Python'
>>> 'Py' * 2
'PyPy'
>>> 'Ba' + 'na' * 2
'Banana
```

Esempi di concatenamento

```
>>> len('Python')
6
>>> len(['PyPy', 'Jython', 'IronPython']) # di una lista
3
>>> len({'a': 3, 'b': 5}) # di un dizionario
2
```

Funzione len

Operazioni con le stringhe

```
>>> s = 'Python'
>>> s.upper() # metodo upper
'PYTHON'
>>> s.lower() # metodo lower
'python'
```

Metodi

```
>>> raggio = 8.4
>>> area = 3.14 * raggio**2
>>> circ = 2 * 3.14 * raggio
>>> s = "L'area è {}, la circonferenza è {}."
>>> s.format(area, circ)
"L'area è 221.5584, la circonferenza è 52.752.

oppure

>>> s = "L'area è %f, la circonferenza è %f."
>>> s % (area, circ)
"L'area è 221.558400, la circonferenza è 52.752000."
```

Formattazione delle stringhe

Tuple

- tipo *built-in*
  - rappresentare una sequenza immutabile di oggetti, in genere eterogenei

```
>>> t = 'abc', 123, 45.67 # la virgola crea la tupla
>>> t                     # rappresentazione
>>> type(t)
<class 'tuple'>

>>> tp = ('abc', 123, 45.67)
>>> tp == tp
True

>>> t = 'nome',    # la vrigola anche per un solo elemento
>>> tv = ()         # tupla vuota

>>> len(tv)
0
```

Tuple

```
>>> t[0]                          # indexing
'abc'
>>> t[:2]                         # slicing
('abc',123)
>>> 'abc' in t                    # in e not in
True
>>> t + ('xyz', 890)              # concatenazione
('abc', 123, 45.67, 'xyz', 890)
>>> t * 2                         # ripetizione
('abc', 123, 45.67, 'abc', 123, 45.67)
>>> t[0] = 'xyz'                  # non ammesso
Traceback . . .
```


Operazioni con le tuple

Tuple

```
>>> len(('abc', 123, 45.67, 'xyz', 890))
>>> min((4, 1, 7, 5))
>>> max((4, 1, 7, 5))
>>> t = ('a', 'b', 'c', 'b', 'a')
>>> t.index('c')
>>> t.count('c')
>>> t.count('b')
```

Provare le seguenti operazioni

Lista

- tipo *built-in*
  - rappresentare una sequenza mutabile di oggetti, in genere omogenei

```
>>> nums = [0, 1, 2, 3] # nuova lista di 4 elementi
>>> nums
[0, 1, 2, 3]
>>> type(nums)
<class 'list'>
>>> empty = [] # nuova lista vuota
>>> empty
[]
>>> one = ['Python'] # nuova lista con un elemento
>>> one
['Python']
```

Creazione di liste

Liste

```
>>> letters = ['a', 'b', 'c', 'd', 'e']
>>> letters[0]                                  # indexing
'a'
>>> t[1:4]                                      # slicing
['b', 'c', 'd']
>>> 'a' in letters                              # in e not in
True
>>> letters + ['f', 'g', 'h']                   # concatenazione
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
>>> [1, 2, 3] * 3                               # ripetizione
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Operazioni con le liste

Liste

```
>>> letters.append('d')
>>> letters.extend(['e', 'f'])
>>> letters.append(['e', 'f'])
>>> letters.pop()
>>> letters.pop(0)
>>> letters.remove('d')
>>> letters.reverse()
>>> letters[1] = 'x'
>>> del letters[1]
>>> letters.clear()
```

Provare le seguenti operazioni

Liste vs tuple

tuple	liste	
Mutabilità	immutabili	mutabili
Lunghezza	fissa	variabile
Accesso agli elementi avviene tramite	indexing	iterazione
Di solito contengono oggetti	eterogenei	omogenei
Simile in C al tipo di dati	struct	array

Confronto tra tuple e liste

Dizionario

- tipo *built-in*
  - mutabile e non ordinato che contiene elementi (*items*) formati da una chiave (*key*) e un valore (*value*)
  - Implementati tramite *tabelle hash*

```
>>> d = {'a': 1, 'b': 2, 'c': 3}
>>> d
{'c': 3, 'a': 1, 'b': 2}
>>> d['a']
1
>>> d['x']  # ?
>>> 'x' in d
False
>>> d = {20: ['Jack', 'Jane'], 28: ['John', 'Mary']}
>>> d
{28: ['John', 'Mary'], 20: ['Jack', 'Jane']}
```

Creazione e uso dei dizionari

Dizionari

```
>>> d = {'a': 1, 'b': 2, 'c': 3}
>>> d['a'] = 10
>>> d
{'c': 3, 'a': 10, 'b': 2}
>>> d['x'] = 123
>>> d
{'x': 123, 'c': 3, 'a': 10, 'b': 2}
>>> del d['x']
>>> d
{'c': 3, 'a': 10, 'b': 2}
```

Operazioni con i dizionari

Dizionari

Metodo	Descrizione
d.items()	Restituisce gli elementi di d come un insieme di tuple
d.keys()	Restituisce le chiavi di d
d.values()	Restituisce i valori di d
d.get(chiave, default)	Restituisce il valore corrispondente a chiave se presente, altrimenti il valore di default (None se non specificato)
d.pop(chiave, default)	Rimuove e restituisce il valore corrispondente a chiave se presente, altrimenti il valore di default (dà KeyError se non specificato)
d.popitem()	Rimuove e restituisce un elemento arbitrario da d
d.update(d2)	Aggiunge gli elementi del dizionario d2 a quelli di d
d.copy()	Crea e restituisce una copia di d
d.clear()	Rimuove tutti gli elementi di d

Set e frozenset

- tipo *built-in*
  - rappresentare un insieme non ordinato di oggetti unici
  - *set mutabili – frozenset immutabili*

```
>>> nums = {10, 20, 30, 40}  # nuovo set di 4 elementi
>>> nums  # gli elementi del set non sono ordinati
{40, 10, 20, 30}
>>> fnums = frozenset(nums)  # nuovo frozenset a partire
dal set nums
>>> fnums
frozenset({40, 10, 20, 30})
>>> {'Python'}  # set di 1 elemento (una stringa)
{'Python'}
>>> empty = set()  # nuovo set vuoto
>>> empty
set()
```

Creazione e uso dei set

Set e frozenset

```
>>> {1, 2, 3, 2, 1}  # i duplicati vengono rimossi
{1, 2, 3}
>>> set('abracadabra')
{'d', 'b', 'a', 'c', 'r'}
>>> frozenset('abracadabra')
frozenset({'d', 'b', 'a', 'c', 'r'})
>>> {'a', 1, (3, 14)}
{1, 'a', (3, 14)}
>>> {'a', 1, (3, 14), [3, 2, 1]}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

Operazioni con i set