Tecnologie Web: Introduzione a Python

- Lezione introduttiva a Python 3.x focalizzata sull'uso del linguaggio nell'ambito delle tecnologie web.

- **Destinatari:** studenti iscritti a un Corso di Laurea triennale in Informatica.

- **Insegnamento:**
  Tecnologie Web (tipicamente secondo o terzo anno, obbligatorio o a scelta).
  Si considerano già trattati gli argomenti: "web come documento ipermediale", paradigma Client/Server, HTTP, HTML, CSS, JavaScript, JQuery, Progressive Web App.
  Si considerano già introdotti aspetti relativi allo sviluppo lato server.

- **Tipologia:** lezione frontale (in presenza, a distanza, o mista) con esempi esplicativi, materiale didattico online (Moodle, Github).
  La lezione sarà supportata da una successiva attività laboratoriale con tutor.

- **Durata:** 45 minuti.

- **Altri ambiti** (riferito all'introduzione a Python)**:**
  Introduzione agli algoritmi e alle strutture dati; strumento nell'ambito di corsi (tipicamente magistrali) di Cloud Computing o Distributed Computing; Internet of Things; corsi pratici di Data Science e Machine Learning (anche in un contesto digital humanities); introduzione alla programmazione per corsi di studio STEM orientati alle scienze computazionali.

Organizzazione didattica

- La lezione è organizzata idealmente in 3 parti.

- **Parte prima: Introduzione** (10 minuti)
  Breve riassunto degli argomenti già coperti dal corso in modo da contestualizzare quanto verrà esposto nel resto della lezione.
  Si richiama l'architettura di un'applicazione web progettata secondo i criteri attualmente considerati canonici e si presenta il Python come linguaggio scelto per lo scripting lato server.

- **Parte seconda: Il linguaggio** (25 minuti)
  È introdotto il linguaggio Python come tale e senza riferimenti espliciti alle tecnologie web.
  Sono fatti frequenti confronti con il C, di cui si assume l'audience abbia comprensione, in modo da evidenziare similitudini e differenze.

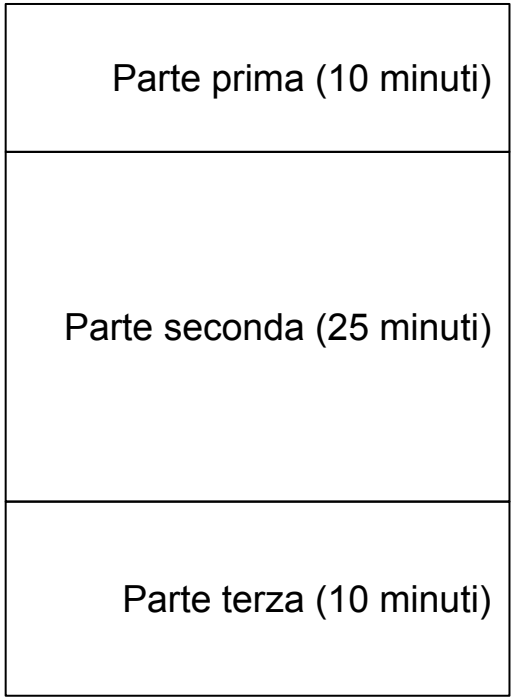
- **Parte terza: Il microframework Flask** (10 minuti)
  Si mostra come realizzare una API minimale con quanto appreso nella parte precedente della lezione.
  Gli argomenti presentati in questa terza parte saranno approfonditi in lezioni successive in cui Python sarà considerato strumento e non argomento di studio.

Sommario

- Introduzione

- Il linguaggio
- Variabili e tipi di dato
- Strutture dati
- Controllo del flusso
- Funzioni e moduli

- Il microframework Flask
- Conclusioni



Tecnologie Web: Introduzione a Python

Raffaele Montella
raffaele.montella@uniparthenope.it

Tecnologie Web

recap

Sviluppo Server Side

Integrazione

Architettura, Protocolli, Linguaggi a Marcatori, Aspetto	Scripting Client Side	Strumenti Server Side

- Il protocollo HTTP è stato progettato per la fruizione di contenuti topologicamente organizzati in ipertesto.
- HTML è un linguaggio a marcatori per la descrizione del documento ipertestuale.
- CSS consente di definire la resa grafica del documento HTML in relazione a dispositivi differenti.

Tecnologie Web

recap

Sviluppo Server Side

Integrazione

Architettura, Protocolli, Linguaggi a Marcatori, Aspetto	Scripting Client Side	Strumenti Server Side

- Il JavaScript consente l'implementazione di applicazioni web dinamiche lato client.
- L'uso del JavaScript è facilitato mediante librerie ausiliarie come jQuery.
- jQuery semplifica e standardizza l'aggiornamento asincrono di porzioni di documento HTML.

- Le Progressive Web App avvicinano le applicazioni web lato client alle applicazioni native del dispositivo in uso.

Tecnologie Web

recap

Sviluppo Server Side

Integrazione

Architettura, Protocolli, Linguaggi a Marcatori, Aspetto

Scripting Client Side

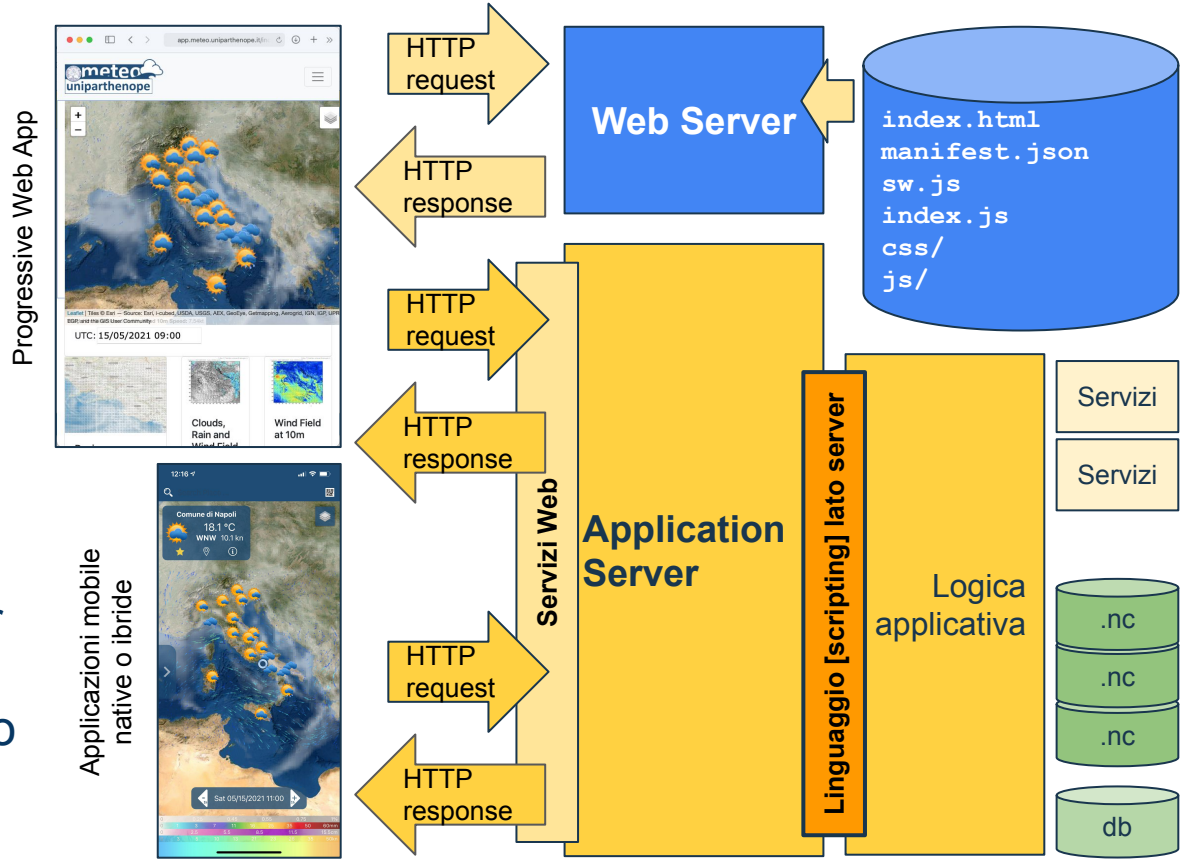
Strumenti Server Side

- Risposte HTTP create dinamicamente in relazione alle richieste HTTP ricevute.
- Pagine HTML, dati in formato XML/JSON, immagini, file.
- Tecnologie varie e differenti (CGI, ASP, ASP.NET, J2EE, PhP, Python, …)
- Mediazione e integrazione con altri servizi (autenticazione, database, IoT, A.I., …)

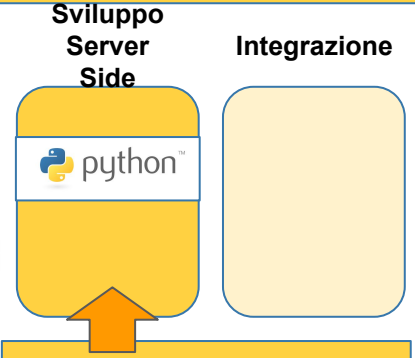
- Tecnologie finalizzate alla scalabilità dell'applicazione.

Architettura di un’applicazione web

- La PWA è servita staticamente dal Web Server (UX).
- I contenuti sono richiesti all’Application Server tramite l’invocazione di Servizi Web (Web API).
- Il comportamento dell’applicazione lato server (BL) è implementato attraverso un linguaggio lato server.



Tecnologie Web & Python



Strumenti Server Side

- Perchè Python?
  - Ha superato Java nel TIOBE Index (Maggio 2020/2021).
  - È semplice da imparare, è facilmente estendibile, è open-source.

May 2021	May 2020	Change	Programming Language	Ratings	Change
1	1		C	13.38%	-3.68%
2	3	▲	Python	11.87%	+2.75%
3	2	▼	Java	11.74%	-4.54%
4	4		C++	7.81%	+1.69%

Quali i fattori del fenomeno conosciuto come “***Escape from Java***”?

Sommario

- Introduzione

- **Il linguaggio**
- Variabili e tipi di dato
- Strutture dati
- Controllo del flusso
- Funzioni e moduli

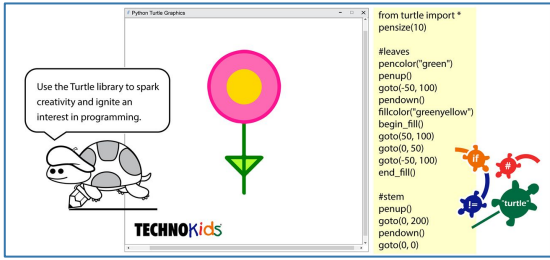
- Microframework Flask
- Conclusioni

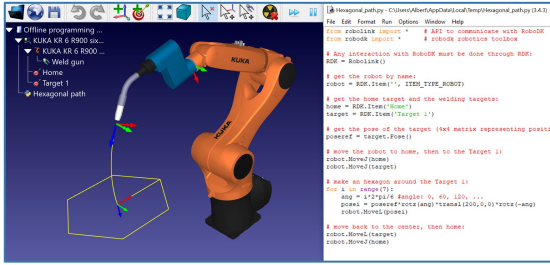
Introduzione

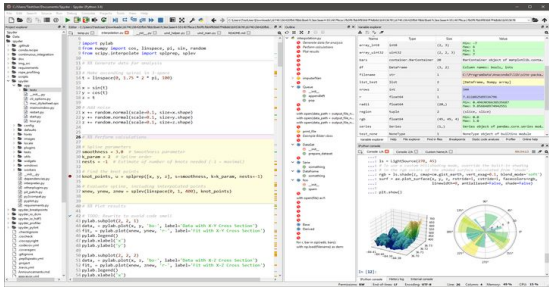
- **Python è un linguaggio di programmazione ad alto livello:**
  adatto a fornire strumenti che permettano di affrontare problematiche complesse.

- **Python è utilizzato in ambiti diversi:**
  dall'insegnamento dei rudimenti della programmazione ai bambini in età scolare, alle scienze computazionali, alla robotica e all'intelligenza artificiale.

- Concepito da **Guido van Rossum** alla fine degli anni '80, è stato rilasciato nel '91 (0.9). La versione 2.0 è del 2000. A partire dal 2008 è disponibile la versione **3.x**.







Caratteristiche

- Linguaggio di programmazione **multi-paradigma**.
- Linguaggio **interpretato**.
- Linguaggio **interattivo**.
- I **tipi di dato** sono gestiti **dinamicamente**.
- È dotato di **garbage collector**.

procedurale

fib.py

```
def fib(n):
    if n<=1:
        return 1
    else:
        return fib(n-1)+fib(n-2)

print(fib(24))
```

oggetti

person.py

```
class Person:
    def __init__(self, f, l):
        self.first=f
        self.last=l

person=Person("Jon","Snow")
```

```
$ cat > helloworld.py << EOF
> print("Hello Python World!")
> EOF
$ python helloworld.py
Hello Python World!
```

```
$ python3
Python 3.6.6 (v3.6.6:4cf1f54eb7, Jun 26 2018, 19:50:54)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 3/5
```

Sintassi - Blocco di codice

- **Python non è “C-like”.**
- **È case sensitive.**
- **È “semi-posizionale”:**
  - La posizione delle istruzioni è parte della sintassi.
  - Il terminatore di linea identifica la fine dell'istruzione.

fact.c

```c
int fact(int n) {
    if (n==0) {
        return 1;
    } else {
        return n*fact(n-1);
    }
}
```

fact.py

```python
def fact(n):
    if n==0:
        return 1
    else:
        return n*fact(n-1)
```

Un blocco di codice è:

- Introdotto dal simbolo **:** (due punti).
- Determinato dall’indentazione (spazi|tab).
- Terminato da una riga vuota.

- Può essere vuoto, ma in tal caso è necessario usare la parola chiave **pass**.

Sintassi - Blocco Globale (global scope)

- È formato dalle linee di codice di uno script che non appartengono ad altri blocchi di codice.
- È il main di default.
- Le righe di codice presenti nel blocco globale sono eseguite in sequenza.
- Le variabili definite nel blocco globale sono visibili nei blocchi di codice in esso contenuto.
- I commenti alla singola linea sono indicati da **#**, multilinea con **'''** … **'''**

global_scope.py

```
# Global scope
name=input("Insert your name:")
print("Your number is",name)
```

main.py

```
# "main"
def main():
    pass

if __name__ == "__main__":
    main()
```

scopes.py

```
# Scopes
a="message a"

def A():
    print("a:",a)
    b="message b"

A()
print("b:",b)
```

Sommario

- Introduzione

- Il linguaggio
- **Variabili e tipi di dato**
- Strutture dati
- Controllo del flusso
- Funzioni e moduli

- Microframework Flask
- Conclusioni

Variabili e tipi di dato

- Le variabili **non** sono **tipizzate**.
- **Identificatori:** lettere maiuscole, minuscole e numeri. Possono iniziare con underscore.
- Le variabili sono **create al primo utilizzo** dell'operatore di assegnazione **=**
- La funzione incorporata **type** restituisce il tipo di dato.
- Il valore **None** non appartiene ad alcun tipo.

variables_1.py

```
# variables 1
m="this is a message"
a=3
pi=3.1415
print(type(m),type(a),type(pi))
```

```
<class 'str'> <class 'int'> <class
'float'>
```

variables_2.py

```
# variables 2
a=4
b=2
c=a+b
d=a/b
print("c:",type(c),"d",type(d))
```

```
c: <class 'int'> d <class 'float'>
```

Variabili e tipi di dato

- Una variabile può cambiare tipo durante l'esecuzione del programma.
- I tipi numerici possono essere **int** (unbounded) o **float** (16 cifre significative).
- Altri tipi di dato:
  - **str**: stringhe
  - **bool**: booleani
  - **complex**: complessi
- Tutti i tipi di dato sono considerati *classi*.

variables_3.py

```
# variables 3
a="abc"
print("a:",a,type(a))
a=3
print("a:",a,type(a))
```

```
a: abc <class 'str'>
a: 3 <class 'int'>
```

variables_4.py

```
# variables 4
a="abc"
b=1
c=1.5
d=True
e=3+5j
print("a:",type(a),"b",type(b),"c:",type(c)
,"d",type(d),"e",type(e))
```

```
a: <class 'str'> b <class 'int'> c: <class
'float'> d: <class 'bool'> e: <class
'complex'>
```

Variabili e tipi di dato - operatori

Assegnazione	= += -= *= /= %= //= **= &= |= ^= >>= <<=
Aritmetici	+ - * / % // (divisione intera) ** (potenza)
Relazionali	< <= > >= == !=
Logici	and or not
Bit a bit	& | ^ ~ << >>

- Gli operatori di assegnazione, aritmetici, relazionali, logici e bit a bit non sono troppo dissimili da altri linguaggi.

Variabili e tipi di dato - operatori

Identità	**is** **is not**	Restituisce True se ambo le variabili si riferiscono allo stesso oggetto. Restituisce True se le due variabili non si riferiscono allo stesso oggetto.
Appartenenza	**in** **not in**	Restituisce True se una sequenza è presente in un oggetto. Restituisce True se una sequenza non è presente in un oggetto.

- Gli operatori di identità e di appartenenza sono caratteristici in Python.

Variabili e tipi di dato - assegnazione multipla

- È possibile effettuare più assegnazioni con un'unica istruzione.
- L'assegnazione è eseguita in base all'ordine.
- L'assegnazione multipla permette di scambiare (swap) i valori di due variabili.

In Python i valori di due variabili possono essere scambiati senza la tradizionale variabile di appoggio. Come mai? Come funziona la tipizzazione dinamica?

variables_5.py

```
# variables 5
a, b, c, d, e, f = 4, 8, 15, 16, 23, 42
print("a:",a,"f:",f)
```

```
a: 4 f: 42
```

swap.py

```
# swap
x = 3
y = 5
print("x:",x,"y:",y)
x, y = y, x
print("x:",x,"y:",y)
```

```
x: 3 y: 5
x: 5 y: 3
```

Variabili e tipi di dato - stringhe

- Una stringa è una lista ordinata di caratteri.
- Le stringhe non sono limitate in lunghezza.
- Il qualificatore di stringa sono le apici singole **' '** o doppie **" "**.
- La funzione **len** restituisce la dimensione della stringa in caratteri.
- È possibile accedere all'i-esimo carattere tramite un indice a base 0.

strings_1.py

```
# strings 1
message = "Hello Python World!"
print(message)
```

```
Hello Python World!
```

strings_2.py

```
# strings 2
s = "Ciao!"
print("s:",s,"len(s):",len(s))
print("s[0]:",s[0],"s[4]:",s[4])
```

```
s: Ciao! len(s): 5
s[0]: C s[4]: !
```

strings_3.py

```
# strings 3
s1 = "abc"; s2="defghijkl"; s=s1+s2
print(s)
```

```
abcdefghijkl
```

Variabili e tipi di dato - stringhe

- È possibile estrarre una sottostringa specificando un range di indici secondo la sintassi: **s[inizio:fine]**
- Gli indici di inizio o fine possono essere assenti.
- Gli indici possono essere negativi (indice con base l'ultimo carattere della stringa).
- Le stringe sono invarianti. Non è possibile assegnare l'i-esimo carattere.

strings_4.py

```
# strings 4
s = "abcdefghijkl"
print(s[2:7])
```

```
cdefg
```

strings_5.py

```
# strings 5
s = "abcdefghijkl"
print("s[:7]:",s[:7])
print("s[2:]:",s[2:])
```

```
s[:7]: abcdefg
s[2:]: cdefghijkl
```

strings_6.py

```
# strings 6
s = "abcdefghijkl"
print("s[:-2]:",s[:-2],"s[-7:]:",s[-7:])
```

```
s[:-2]: abcdefghij s[-7:]: fghijkl
```

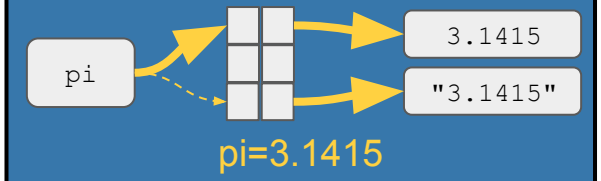
Variabili e tipi di dato - classi e oggetti

- Tutti i dati prendono il nome generico di **oggetto.**
- Un **oggetto** può essere visto come un'area di memoria in cui sono conservati **valori** e su cui sono definite le **operazioni** che è possibile eseguire su di essi.
- Esistono **oggetti predefiniti** (variabili, liste, stringhe, tuple ed altri) e **oggetti definiti dall'utente**.
- Su ogni tipo di oggetto sono definite particolari operazioni chiamate **metodi**.
- I metodi hanno la forma:

```
result = object_name.method_name(args)
```

Tipizzazione dinamica

- Le variabili sono rappresentate da istanze di oggetti.
- I valori (tipi di dato base, strutture dati) sono rappresentati da istanze di oggetti.
- L'associazione variabile/valore è fatta attraverso i puntatori.
- In questo modo l'oggetto dato e l'oggetto variabile sono disaccoppiati.



pi=3.1415

Variabili e tipi di dato - classi e oggetti

- Le classi definite dallo sviluppatore sono introdotte dalla parola chiave **class** seguita dal nome della classe e da **:** (due punti) che dà inizio al blocco di codice.
- La parola chiave **self** è il riferimento all'istanza dell'oggetto corrente (ricorda il **this** di C++/Java).
- Gli attributi sono definiti in maniera implicita.

geopoint.py

```
# geopoint
class GeoPoint:
    def __init__(self, lat, lon):
        self.__lat=lat
        self.__lon=lon

    def getLat(self):
        return self.__lat

    def getLon(self):
        return self.__lon

pos1=GeoPoint(40.85,14.28)
print(pos1.getLat(), pos1.getLon())
```

```
40.85 14.28
```

Variabili e tipi di dato - classi e oggetti

- Attributi e metodi sono pubblici di default a meno che non siano postfissi __ (doppio underscore) al nome.
- I metodi sono definiti come *funzioni* all'interno del blocco di codice **class**
- Tutti i metodi di istanza devono avere **self** come primo parametro (che non sarà necessario quando invocati).
- Alcuni metodi hanno funzioni speciali come **__init__** che è il metodo costruttore.

geopoint.py

```
# geopoint
class GeoPoint:
    def __init__(self, lat, lon):
        self.__lat=lat
        self.__lon=lon

    def getLat(self):
        return self.__lat

    def getLon(self):
        return self.__lon

pos1=GeoPoint(40.85,14.28)
print(pos1.getLat(), pos1.getLon())
```

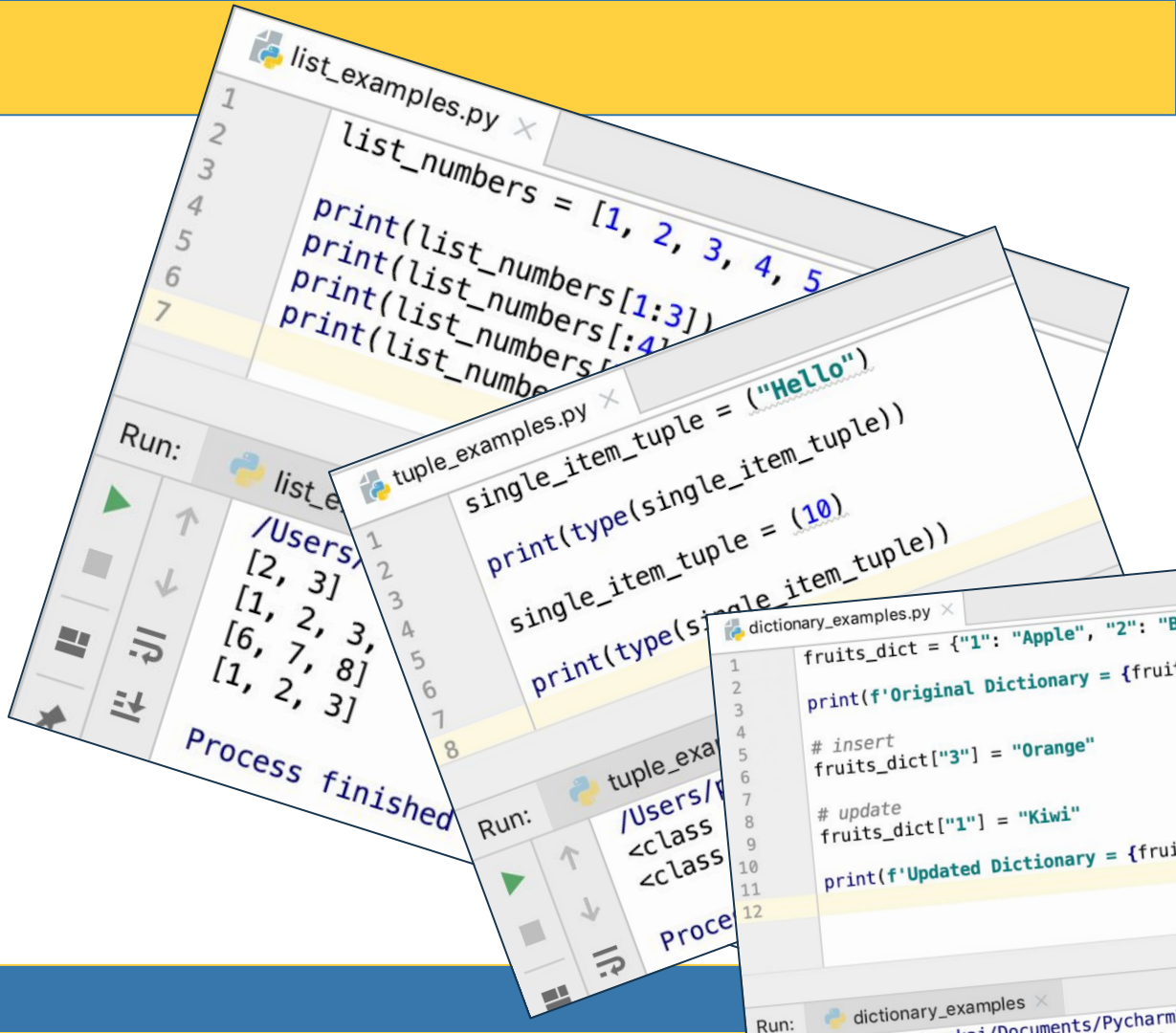
```
40.85 14.28
```

Sommario

- Introduzione

- Il linguaggio
- Variabili e tipi di dato
- **Strutture dati**
- Controllo del flusso
- Funzioni e moduli

- Microframework Flask
- Conclusioni



Strutture dati - liste - `list()`

- Una **list** è un tipo di dato usato per raggruppare valori, non necessariamente dello stesso tipo.
- Una list contiene elementi separati da **virgole** e racchiusi da parentesi **quadre**.
- Non va confusa con l'array (per il quale esiste un modulo dedicato).

lists_1.py

```
# lists 1
items = ["apple",3.14,True]
print(len(items))
```

```
3
```

lists_2.py

```
# lists 2
fruits = ["apple","banana","raspberry","orange"]
print(len(fruits), fruits[2])
```

```
4 raspberry
```

lists_3.py

```
# lists 3
avengers = []
avengers.append("Iron Man")
avengers.append("Cap. America")
avengers.append("Black Widow")
print(avengers)
```

```
['Iron Man', 'Cap. America', 'Black Widow']
```

Strutture dati - `list()`

- Possono essere concatenate.
- È possibile accedere al singolo elemento, come con le stringhe.
- È variante: è possibile cambiare il valore dell'i-esimo elemento.
- Le liste possono essere annidate (liste di liste).

lists_4.py

```
# lists 4
f1 = ["apple","banana","raspberry"]
f2 = ["passionfruit","pineapple"]
fruits = f1 + f2
print(len(fruits), fruits)
```

```
5 ['apple', 'banana', 'raspberry',
'passionfruit', 'pineapple']
```

lists_5.py

```
# lists 5
l = [4, 8, 15, 16, 23, 42]
print("l[0]",l[0],"l[3:5]",l[3:5],"l[-1]",l[-1])
```

```
l[0] 4 l[3:5] [16, 23] l[-1] 42
```

lists_6.py

```
# lists 6
l = ["abc","def","ghi","jkl"];l[1]="xxx"
print("l:",l)
```

```
l: ['abc', 'xxx', 'ghi', 'jkl']
```

Strutture dati - `tuple()`

- Una **tupla** è una sequenza di dati, simile alla list.
- Consiste di un elenco di valori separati da **virgole** e racchiuse tra parentesi **tonde**.
- A differenza delle list, le tuple sono invarianti: gli elementi **non possono essere cambiati**.

tuple_1.py

```
# tuple 1
pos = (14.28, 40.85)
print(len(pos), type(pos))
```

```
2 <class 'tuple'>
```

tuple_2.py

```
# tuple 2
t = (4, 8, 15, 16, 23, 42)
x = (8, 2)
s = t+x
print("s:",s)
```

```
s: (4, 8, 15, 16, 23, 42, 8, 2)
```

tuple_3.py

```
# tuple 3
t = ("abc","def","ghi")
print("def?t:","def" in t,"jkl?t:","jkl" in t)
```

```
def?t: True jkl?t: False
```

Strutture dati - dizionari - `dict()`

- **dictionary** è una struttura dati che consente di memorizzare coppie chiave:valore.
- Chiave e valore possono essere di qualsiasi tipo senza alcuna limitazione.
- I dizionari possono essere annidati.
- Sintassi:
  - definiti attraverso l'uso di **{** e **}**
  - Le coppie chiave:valore sono separate da **:**
  - Notazione **JSON**.
- La chiave può essere usata come indice per l'accesso al valore dizionario.

dict_1.py

```
# dict 1
person = {
    "first":"Jon",
    "last":"Snow"
}
print(len(person), person)
```

```
2 {'first': 'Jon', 'last':
'Snow'}
```

dict_2.py

```
# dict 2
person = { "first":"Jon" }
person["last"]="Snow"
print(type(person), person)
print("f:",person["first"])
print("l:",person["last"])
```

```
<class 'dict'> {'first':
'Jon', 'last': 'Snow'}
f: Jon
l: Snow
```

Strutture dati - dizionari - `dict()`

dict_3.py

```
# dict 3
course = {}
course["id"]="TW6"
course["isElective"]=True
course["topics"]=["C/S","HTTP","HTML","CSS","JavaScript","jQuery","PWA","Python"]
course["taughtBy"]={"first":"Raffaele","last":"Montella"}
print(course)
print(len(course),len(course["topics"]),len(course["taughtBy"]))
```

```
{'id': 'TW6', 'isElective': True, 'topics': ['C/S', 'HTTP', 'HTML', 'CSS',
'JavaScript', 'jQuery', 'PWA', 'Python'], 'taughtBy': {'first': 'Raffaele',
'last': 'Montella'}}
4 8 2
```

- Combinando opportunamente liste, tuple e dizionari è possibile gestire strutture dati complesse.
- Per rimuovere una voce del dizionario si usa la parola chiave **del**

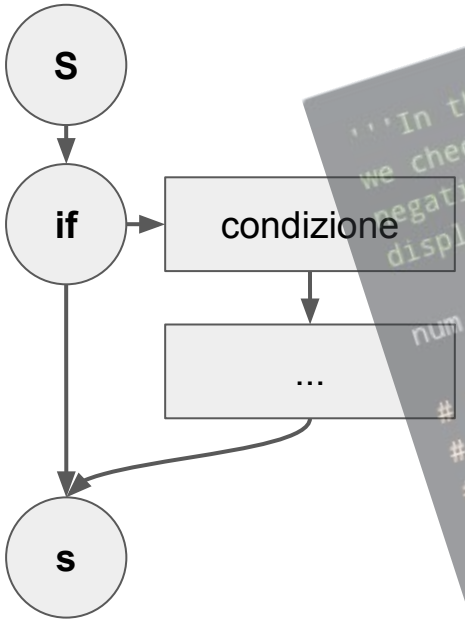
  ```
  del course["isElective"]
  ```

Sommario

- Introduzione

- Il linguaggio
- Variabili e tipi di dato
- Strutture dati
- **Controllo del flusso**
- Funzioni e moduli

- Microframework Flask
- Conclusioni




```
'''In this program,
we check if the number is positive or
negative or zero and
display an appropriate message'''

num = 3.4

# Try these two variations as well:
# num = 0
# num = -4.5

if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

Controllo del flusso - costrutto condizionale

- Blocco di selezione a tre vie: **if/elif/else**.
- **Operatore ternario:** variabile = valore1 **if** condizione **else** valore2
- Python, a differenza di molti linguaggi C-like, non ha un costrutto di tipo switch.

branching_1.py

```
# branching 1
import random
x = random.randint(1,100)
if x<33:
    print("1st 3rd")
elif 33 <= x <= 66:
    print("2nd 3rd")
else:
    print("3rd 3rd")
```

```
2nd 3rd
```

branching_2.py

```
# branching 2
import random
a = random.randint(1,100)
b = random.randint(1,100)
m = a if a>0 else b
print("a:",a,"b:",b,"max:",m)
```

```
a: 32 b: 24 max: 32
```

Controllo del flusso - ciclo while

- Ciclo iterativo con test in ingresso (**while**).
- L'istruzione **break** interrompe il ciclo.
- L'istruzione **continue** interrompe l'iterazione corrente e passa a quella successiva.
- Python, a differenza di molti linguaggi C-like, non ha un ciclo iterativo con test in uscita.

cycles_1.py

```
# cycles 1
x=0
while x<100:
    print("x:",x)
    x=x+1
```

```
x: 0
x: 1
...
x: 99
```

cycles_2.py

```
# cycles 2
import random
while True:
    x=random.randint(1,100)
    if x==50:
        break
print("x:",x)
```

```
x: 50
```

Controllo del flusso - ciclo for

- Ciclo iterativo a numero di iterazioni prefissato (**for**).
- A differenza di altri linguaggi, il ciclo **for** esplora una collezione di dati iterabile.
- La funzione incorporata **range** restituisce una lista di numeri, dato un intervallo e un incremento.

cycles_3.py

```
# cycles 3
for x in range(100):
    print("x:",x)
```

```
x: 0
x: 1
...
x: 99
```

cycles_4.py

```
# cycles 4
import random
s=""
for x in range(33,66,3):
    s=s+str(x)+" "
print("s:",s)
```

```
s: 33 36 39 42 45 48 51 54 57 60 63
```

Sommario

- Introduzione

- Il linguaggio
- Variabili e tipi di dato
- Strutture dati
- Controllo del flusso
- **Funzioni e moduli**

- Microframework Flask
- Conclusioni

Funzioni e moduli

- Le funzioni predefinite (incorporate o built-in) di Python sono più di 60.

`abs()`	restituisce il valore assoluto di un numero
`input()`	visualizza un testo, attende che l'utente inserisca da tastiera una serie di caratteri e prema il tasto Invio; restituisce la stringa inserita (da convertire eventualmente in un numero)
`eval()`	valuta dinamicamente un'espressione fornita come stringa
`float()`	trasforma l'argomento in un numero reale
`int()`	trasforma l'argomento in un intero
`round()`	arrotonda all'intero più vicino
`max()` `min()`	restituiscono il massimo e il minimo in una lista di valori separati da virgole
`print()`	stampa a video il contenuto inserito all'interno delle parentesi
`type()`	restituisce il tipo di un dato
`range(n)`	restituisce un elenco di n valori da 0 a n-1
`range(i,f,s)`	restituisce un elenco di valori da i(incluso) a f(escluso) step s

Funzioni e moduli - passaggio di parametri

- Una funzione è definita attraverso la parola chiave **def**.
- Una funzione può avere nessuno, uno o n parametri.
- I parametri possono avere valori di default.
- I parametri possono essere passati come coppie chiave/valore.
- Una funzione può restituire un risultato usando la parola chiave **return**.
- I parametri sono passati per valore.

funcs_1.py

```
# funcs 1
def somma(a, b):
    return a+b

print("s:",somma(10,9))
```

```
s: 19
```

funcs_2.py

```
# funcs 2
x=7
def inc(a):
    a=a+1
    return a

print("x:",x)
i=inc(x)
print("x:",x,"i:",i)
```

```
x: 7
x: 7 i: 8
```

Funzioni e moduli - i moduli

- Funzioni, variabili e framework di classi possono essere raggruppati in moduli.
- Un modulo in Python è definito attraverso un file o una directory il cui contenuto è opportunamente organizzato.
- L'uso dei moduli consente di evitare ambiguità grazie alla definizione di **namespace** in cui un identificativo ha valore.
- È possibile importare un intero modulo o solo parte di esso.

mods_1.py

```
# mods 1
import random

print(random.randint(1,10))
```

```
7
```

mods_2.py

```
# mods 2
from random import randint

print(randint(1,10))
```

```
5
```

mods_3.py

```
# mods 3
from flask import Flask

app = Flask(__name__)
```

```
ModuleNotFoundError: No
module named 'flask'
```

Funzioni e moduli - i moduli

- Perchè un modulo possa essere importato, questo deve essere raggiungibile tramite il percorso di ricerca contenuto nella variabile di ambiente PYTHONPATH.
- I moduli sono spesso componenti software da installare separatamente.
- I moduli possono implementare complesse funzionalità il cui uso è semplificato dall'interfaccia Python (library wrap).
- Esistono varie metodologie per l'installazione di moduli Python.
- Il Package Installer for Python consente di installare moduli dal Python Package Index.

Funzioni e moduli - i moduli

- I moduli possono essere installati:
  - Per tutti gli utenti (system)
  - Per l'utente (user)
  - Ambiente virtuale (venv)
- È preferibile l'installazione di un "ambiente virtuale" in modo da limitare la visibilità dei moduli installati alla sola applicazione corrente.
- Esempio: modulo per il calcolo del codice fiscale italiano.

```
$ python3 -m venv venv
$ . venv/bin/activate
(venv)$ pip install python-codicefiscale
```

```
Collecting python-codicefiscale
Downloading
https://files.pythonhosted.org/packages/95
/65/f6be1cc7a32c77e6478d61a48b15ec700331f7
5186f08977e538fc5b2abb/python-codicefiscal
e-0.3.7.tar.gz (131kB)
...
Running setup.py install for
python-codicefiscale ... done
Successfully installed
python-codicefiscale-0.3.7
python-dateutil-2.8.1 ...
```

Funzioni e moduli - i moduli

- L'esempio riportato consente di calcolare il codice fiscale italiano.
- I moduli consentono di accrescere le funzionalità di Python praticamente senza limiti.
- Alcuni moduli sono "**pure Python**", altri necessitano di una fase di building gestita automaticamente dal gestore dei pacchetti.
- Il modulo **flask** incapsula un microframework per la produzione di **contenuti web dinamici lato server**.

mods_4.py

```
# mods 4
from codicefiscale import codicefiscale

person={
     "surname":'Montella',
     "name":'Raffaele',
     "sex":'M',
     "birthdate":'10/05/1972',
     "birthplace":'Napoli'
}

result = codicefiscale.encode(
     surname=person["surname"],
     name=person["name"],
     sex=person["sex"],
     birthdate=person["birthdate"],
     birthplace=person["birthplace"])

print(result)
```

```
MNTRFL72E10F839I
```

Sommario

- Introduzione

- Il linguaggio
- Variabili e tipi di dato
- Strutture dati
- Controllo del flusso
- Funzioni e moduli

- **Microframework Flask**
- Conclusioni

Microframework Flask - introduzione

- Flask è un ***microframework*** per Python basato su Werkzeug e usato per sviluppare applicazioni web e Jinja 2.
- È concesso in licenza BSD.
- Offre le informazioni fondamentali per il routing dell'URL e il rendering della pagina.
- Flask offre suggerimenti, ma non applica alcuna dipendenza o layout del progetto.

Microframework Flask - introduzione

- Spetta allo sviluppatore scegliere gli strumenti e le librerie che vuole usare.
- Ci sono molte estensioni fornite dalla comunità che rendono facile l'aggiunta di nuove funzionalità.
- Flask è definito un framework "micro" perché non mette direttamente a disposizione funzionalità come la convalida del modulo, l'astrazione di database, l'autenticazione e così via.

Microframework Flask - dipendenze

- **Werkzeug:** Implementa WSGI, l'interfaccia standard di Python tra applicazioni e server web.
- **Jinja2:** È un template language che esegue il rendering delle pagine utilizzate dall'applicazione.
- **MarkupSafe:** Rende sicuro il rendering dei template effettuato con Jinja2.
- **ItsDangerous:** Usato per proteggere il cookie di sessione di Flask.
- **Click:** Framework per la gestione delle applicazioni da linea di comando.

Microframework Flask - installazione

```
$ python3 -m venv venv
$ . venv/bin/activate
(venv)$ pip install Flask
```

```
Collecting Flask
Using cached
https://files.pythonhosted.org/packages/bf/73/9180d22a40da68382e9cb6edb66a74bf09
cb72ac825c130dce9c5a44198d/Flask-2.0.0-py3-none-any.whl
...
Installing collected packages: MarkupSafe, Jinja2, click, itsdangerous,
dataclasses, Werkzeug, Flask
Successfully installed Flask-2.0.0 Jinja2-3.0.0 MarkupSafe-2.0.0 Werkzeug-2.0.0
click-8.0.0 dataclasses-0.8 itsdangerous-2.0.0
```

- Flask può essere installato mediante il Package Installer for Python (pip).
- È sempre conveniente utilizzare un ambiente locale all'applicazione.

Microframework Flask - Hello World

- Flask permette di definire il routing delle risorse web attraverso l'uso delle direttive di decorazione (*decorators*).
- **`@app.route(path)`:** riscrive il codice della funzione seguente (`root()`) in modo che questa sia invocata quando il server riceve una HTTP request di una risorsa identificata dal path.

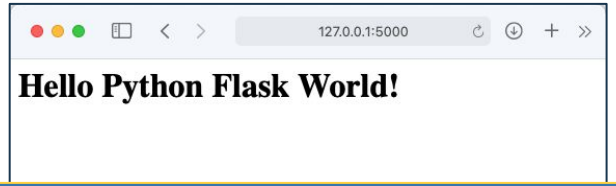
flask_1.py

```
# flask 1
from flask import Flask

app = Flask(__name__)

@app.route("/")
def root():
     html='''
         <html>
           <head>
             <title>Python Flask Hello World</title>
           </head>
           <body>
             <h1>Hello Python Flask World!</h1>
           </body>
         </html>'''
      return html
```

Il funzionamento delle funzioni di decorazione in Python è un argomento avanzato sarà approfondito il laboratorio.

Microframework Flask - Hello World

- Il nome del file che implementa l'applicazione deve essere specificato nella variabile di ambiente FLASK_APP.
- In FLASK_ENV va specificato l'ambiente di esecuzione.
- Ip e porta su cui far eseguire il server vanno specificati sulla linea di comando.




```
$ export FLASK_APP=flask_1.py
$ export FLASK_ENV=debug
$ flask run -h 0.0.0.0 -p 5000
```

```
* Serving Flask app 'flask_1.py' (lazy loading)
* Environment: DEBUG
* Debug mode: off
* Running on all addresses.
WARNING: This is a development server. Do not
use it in a production deployment.
* Running on http://192.168.1.32:5000/ (Press
CTRL+C to quit)
```

```
127.0.0.1 - - [16/May/2021 16:19:43] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [16/May/2021 16:19:43] "GET /favicon.ico HTTP/1.1" 404 -
```

Microframework Flask - Semplice Web API

- Flask consente un raffinato templating HTML tramite Jinja2 e Jade.
- Flask consente di produrre servizi web (Web Application Program Interface).
- L'esempio mostra come rispondere a una HTTP request con un JSON contenente un codice fiscale.

flask_2.py

```
# flask 2
from flask import Flask, jsonify

app = Flask(__name__)

@app.route("/italian-fiscal-code")
def get_italian_fiscal_code():
     result={ "fiscalCode":"MNTRFL72E10F839I" }
     return jsonify(result)
```

Eseguire l'applicazione flask_2.py

```
$ export FLASK_APP=flask_2.py
$ export FLASK_ENV=debug
$ flask run -h 0.0.0.0 -p 5000
```

```
127.0.0.1 - - [16/May/2021 16:43:46]
"GET /italian-fiscal-code HTTP/1.1" 200
-
```

Invocare il servizio tramite un'altra shell

```
$ curl
http://localhost:5000/ita
lian-fiscal-code
```

```
{"fiscalCode":"MNTRFL72E10F839I"}
```

Microframework Flask - Web API Codice Fiscale

- Esempio di uso di Python come "*glue language*": è usato il modulo codicefiscale.
- **`@app.route(path, methods)`**: è specificato che la funzione `italian_fiscal_code` è invocata solo se si ha una HTTP request di tipo POST.
- L'oggetto request del microframework Flask consente di accedere alle coppie chiave/valore inviate con il metodo POST.

flask_3.py

```
# flask 3
from flask import Flask, jsonify, request
from codicefiscale import codicefiscale

app = Flask(__name__)

@app.route("/italian-fiscal-code", methods=["POST"])
def italian_fiscal_code():
  fiscal_code = codicefiscale.encode(
    surname=request.form["surname"],
    name=request.form["name"],
    sex=request.form["sex"],
    birthdate=request.form["birthdate"],
    birthplace=request.form["birthplace"])
  result={ "fiscalCode":fiscal_code }
  return jsonify(result)
```

Microframework Flask - Web API Codice Fiscale

Eseguire l'applicazione flask_3.py

```
$ export FLASK_APP=flask_3.py
$ export FLASK_ENV=debug
$ flask run -h 0.0.0.0 -p 5000
```

```
127.0.0.1 - - [16/May/2021 17:07:25] "POST /italian-fiscal-code HTTP/1.1" 200 -
```

Invocare il servizio tramite un'altra shell

```
$ curl --data
"surname=montella&name=raffaele&sex=M&birthdate=10/05/1972&bi
rthplace=Napoli" http://localhost:5000/italian-fiscal-code
```

```
{"fiscalCode":"MNTRFL72E10F839I"}
```

Per quale motivo è stato specificato che la funzione `italian_fiscal_code` deve essere invocata solo nel caso di richiesta tramite HTTP POST?

- In mancanza di un'applicazione web client progettata per consumare la Web API, si adopera un web client a linea di comando come wget o curl.

Sommario

- Introduzione

- Il linguaggio
- Variabili e tipi di dato
- Strutture dati
- Controllo del flusso
- Funzioni e moduli

- Microframework Flask
- **Conclusioni**

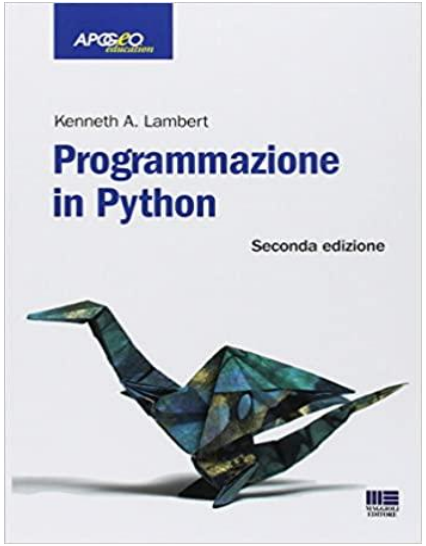
Presente nei sistemi operativi Unix-like (Linux, MacOS, ...), da installare su Windows.

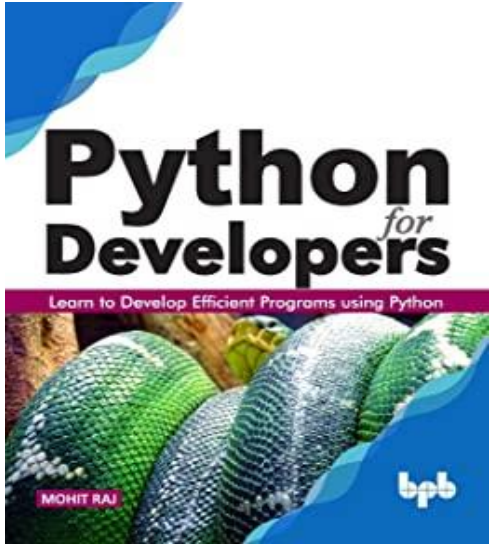
https://python.org

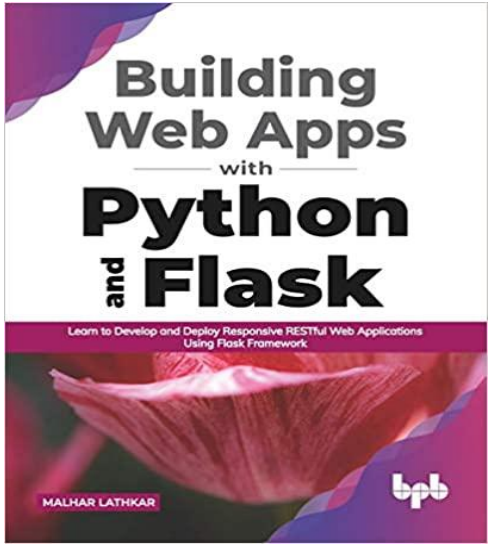
Conclusioni

- In relazione all'architettura di un'applicazione web attuale, è stata motivata la scelta del linguaggio Python come strumento di scripting lato server.
- È stato introdotto il linguaggio considerando note le basi della programmazione. Tuttavia ogni aspetto relativo a Python qui introdotto meriterebbe un approfondimento.
- È stato introdotto il microframework Flask per la creazione di applicazioni web lato server ed è stato usato per implementare delle semplici Web API.
- Le prossime lezioni del corso saranno dedicate ad approfondire la conoscenza di Flask (gestione del routing, metodi HTTP, sessioni, autenticazione, database, servizi web REST).

Approfondimenti







https://www.tutorialspoint.com/python/

https://www.w3schools.com/python/

https://books.goalkicker.com/PythonBook/

https://flask.palletsprojects.com/en/2.0.x/quickstart/

https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world

https://openbookproject.net/thinkcs/python/english3e/

Funzioni e moduli - passaggio di parametri

- Un riferimento ad un oggetto come parametro di una funzione.

funcs_3.py

```
# funcs 3
class Person:
  def __init__(self, f, l):
    self.first=f
    self.last=l

  def print(self):
    print(self.first,self.last)

def swapFirstLast(p):
  p.first,p.last = p.last,p.first

person=Person("Jon","Snow")
person.print()
swapFirstLast(person)
person.print()
```

```
Jon Snow
Snow Jon
```

Funzioni e moduli - bisezione

- Ricerca ricorsiva tramite bisezione.

funcs_n.py

```
# funcs n
import random
data=[]
for x in range(100):
        if random.randint(1,100)<33:
                data.append(x)

def find(a,b,key):
        i=round((a+b)/2)
        if 0 <= i < len(data) and i != a and i != b:
                if data[i]==key:
                        return i
                if data[i]>key:
                        return find(a,i,key)
                else:
                        return find(i,b,key)
        return -1
print("i:",find(0,len(data)-1,50))
```

```
i: 19
```