



Progettazione e sviluppo di applicazioni web

Azienda Speciale di Formazione “Scuola Paolo Borsa”
Ing. Masciadri Andrea <masciadri.andrea@gmail.com>



Oggi

Algoritmi e Programmazione

BREAK

Python

Algoritmi e programmazione



Fasi per progettare un algoritmo

- Formalizzare il problema
- Progettare
- Stendere l'algoritmo
- Verificare

problema

- Problema: Un comune decide di mettere una tassa sul passo carrabile di 9.8 euro al metro quadro. Ogni passo carrabile è stato denunciato definendone le dimensioni in altezza e larghezza.

Calcolare la tassa che si deve pagare per ogni passo carrabile

Formalizzazione problema

- Problema: Un comune decide di mettere una tassa sul passo carrabile di 9.8 euro al metro quadro. Ogni passo carrabile è stato denunciato definendone le dimensioni in altezza e larghezza.

Calcolare la tassa che si deve pagare per ogni passo carrabile

INPUT: altezza e larghezza passo carrabile

OUTPUT: tassa

Raffinamento formalizzazione

Definizione:

INPUT: altezza e larghezza passo carrabile

OUTPUT: tassa

Raffinamento: Il Dialogo Uomo – Computer

Inserisci la lunghezza del passo carrabile:

4

Inserisci la larghezza del passo carrabile:

5

tassa da pagare


196

Progettare

Comincio a definire I passi fondamentali che deve fare l'algoritmo:

- 1.Acquisire I valori in input
- 2.Calcolare la tassa
- 3.Visualizzare la tassa

LA RAPPRESENTAZIONE DEGLI ALGORITMI: DIAGRAMMI DI FLUSSO



La rappresentazione degli algoritmi a diagrammi di flusso (*flowchart*) è essenzialmente grafica. Si tratta di un disegno composto da una serie di costrutti base (o blocchi),
che rappresentano le diverse azioni da compiere, collegati da frecce.

●●● PROVIAMO INSIEME

Dato un numero intero positivo, dire se è pari o dispari

L'algoritmo che risolve il problema proposto è un tipico esempio di algoritmo in cui è necessario prendere una decisione in base a una condizione.

Prima di tutto leggiamo il numero e lo memorizziamo in una variabile *numero*. Come si fa a stabilire se *numero* è pari o dispari? Partiamo dalla definizione matematica: un numero è pari se e solo se è divisibile per 2. Ossia quando il resto della divisione per 2 è uguale a 0. Come si è visto, abbiamo a disposizione un operatore che ci restituisce il resto della divisione tra due numeri: l'operatore modulo. Ci basta quindi calcolare il resto di *numero* diviso 2 e verificare se è uguale a 0 oppure no: nel primo caso, il numero dato è pari, altrimenti è dispari (Fig. 9).

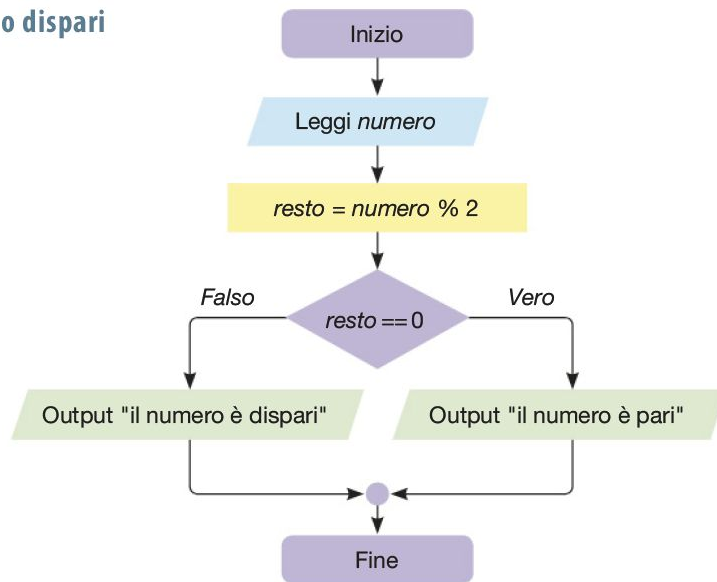
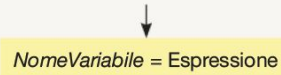


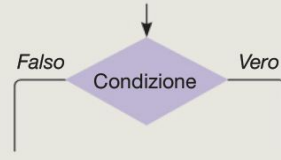
Fig. 9 Diagramma di flusso dell'algoritmo che determina numeri pari e dispari.

Blocchi del flowchart	Descrizione
<pre> graph TD Inizio([Inizio]) --> Fine([Fine]) Fine --> Input[/Input NomeVariabile/] Input --> Output[/Output NomeVariabileDaVisualizzare/] </pre>	<p>Blocco di inizio</p> <ul style="list-style-type: none"> • identifica il punto di ingresso della computazione; • deve essere sempre presente; • è unico per ogni algoritmo.
	<p>Blocco di fine</p> <ul style="list-style-type: none"> • identifica il punto finale della computazione; • deve essere sempre presente altrimenti l'algoritmo non termina; • è unico per ogni algoritmo.
	<p>Blocco di input</p> <ul style="list-style-type: none"> • l'azione attraverso la quale si realizza l'input (per esempio Leggi) deve essere seguita dal nome di una variabile in cui memorizzare il dato in ingresso.
	<p>Blocco di output</p> <ul style="list-style-type: none"> • l'azione attraverso la quale si realizza l'output (per esempio Scrivi) deve essere seguita dal nome di una variabile il cui contenuto deve essere visualizzato.



Blocco di azione

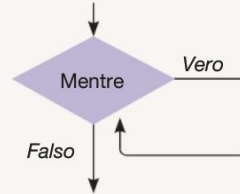
- calcola il valore di un'espressione e assegna il risultato a una variabile.



Blocco di selezione

- verifica un'espressione booleana e si dirama in due direzioni a seconda che la proposizione abbia valore *Vero* o *Falso*.

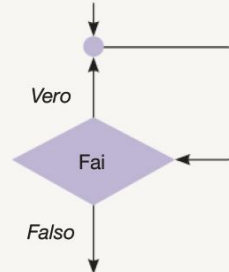
pre-condizionale



Blocco di iterazione

- viene usata per:
 - a. eseguire un certo insieme di istruzioni fino a quando non si verifica un certo evento;
 - b. eseguire un certo insieme di istruzioni un numero prefissato di volte.
- è pre-condizionale se la condizione viene valutata prima di entrare nel ciclo;
- è post-condizionale se la condizione viene valutata dopo aver eseguito almeno una volta le istruzioni del ciclo.

post-condizionale



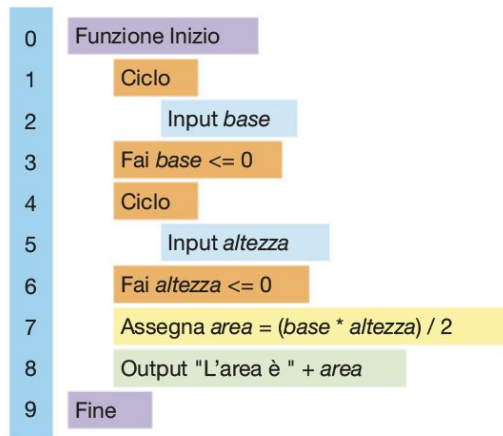
LA RAPPRESENTAZIONE DEGLI ALGORITMI: PSEUDOCODIFICA



Un altro modo per rappresentare gli algoritmi è la **pseudocodifica**. Essa si basa su quello che possiamo chiamare pseudocodice e utilizza parole chiave della lingua corrente, ma con regole costrutti propri dei linguaggi di programmazione.

PROVIAMO INSIEME**Scriviamo l'algoritmo per l'area di un triangolo in pseudocodice**

Qui sotto trovi un esempio. Come puoi facilmente vedere, a ogni blocco del diagramma di flusso corrisponde una pseudoistruzione molto simile.



Nota quella che viene chiamata *indentazione*: ogni volta che delle istruzioni sono annidate all'interno di un blocco vengono spostate tutte a destra. Nell'esempio, tutte le istruzioni del nostro algoritmo sono spostate a destra rispetto ai blocchi di inizio e fine; le istruzioni all'interno del ciclo sono spostate a destra rispetto al blocco Ciclo. In tal modo si comprende subito quali parti dell'algoritmo dipendono da quali altre. Osserva anche che, poiché come si è detto uno pseudocodice gode di una certa flessibilità, esso rimane del tutto corretto se decidiamo di utilizzare per esempio la parola "Stampa" al posto di "Output" per indicare l'istruzione di output.

Python



IDE Installation



IntelliJ IDEA

The most intelligent Java
IDE



CLion

Smart cross-platform IDE
for C and C++



PyCharm

Powerful Python & Django
IDE



PhpStorm

IDE for Web & PHP

PyCharm è un valido IDE.. ma potete scegliere quello che preferite!

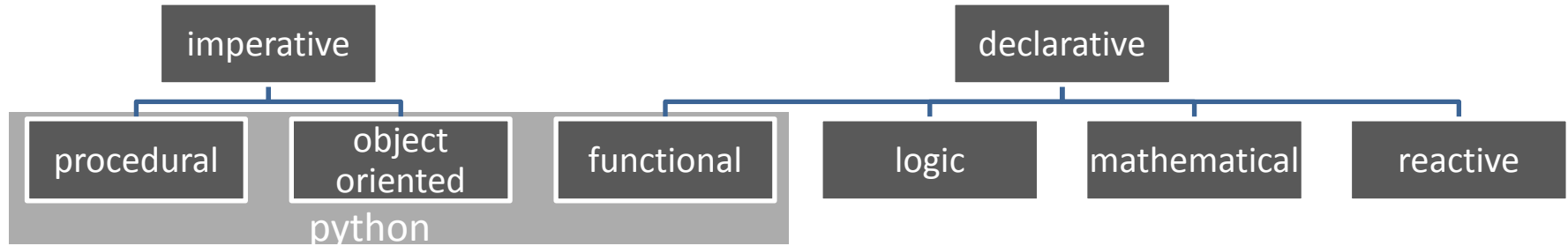


Python VS C

	C	Python
Architecture	Procedural, high-level, general-purpose, compiled programming language	multi-paradigm, (also) interpreted programming language
Variables	requires a compulsory declaration of variable types	loosely-typed Dynamically-typed
Pointers	yes	not supported
Garbage collector	not supported	yes
Indentation	Nice have	mandatory
Built-in functions	limited number of built-in functions	large library of built-in functions



Programming language paradigms





Object oriented programming (OOP)

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods.

Wikipedia

Class

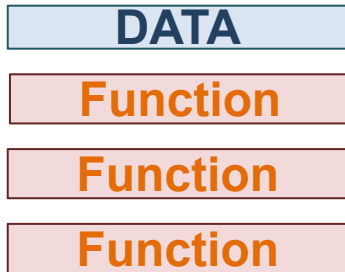
Is a template that defines the behavior of an object, defines the data (attributes) and the functions that operates on the data (methods).

Object

Is an instance of a class, initialized with specific data, objects are created and eventually destroyed at run time and belongs to a specific class.

OOP Core principles

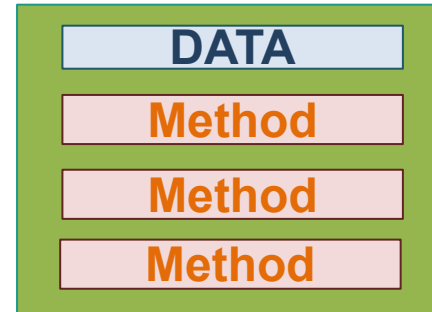
Procedural



- Data is accessible to all the functions of the program

Encapsulation: Bind the data with the code that manipulates it

Object Oriented

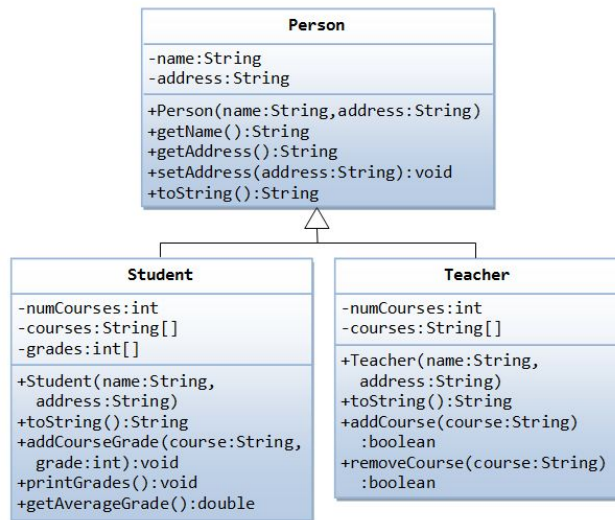


- Data is encapsulated in the object invisible to the world "outside"
- Methods can be used to access and manipulate data

OOP Core principles

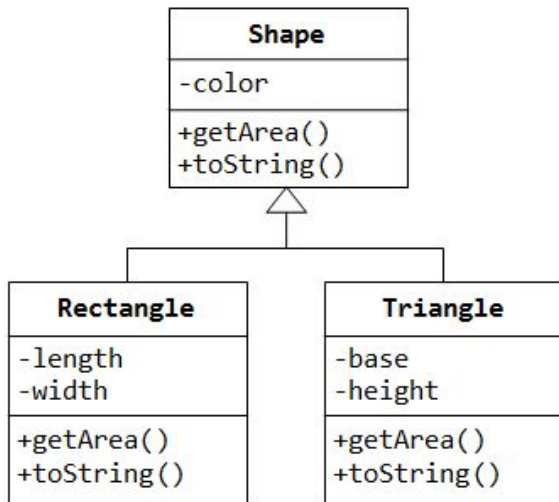
Inheritance: A child

behavior of a parent object



OOP Core principles

Polymorphism: Ability to present the same interface for different underlying form





SYNTAX



Comments

```
# inline comment
```

```
"""
```

```
This is a multiline comment that can be used to  
create automatic documentation for code
```

```
"""
```

```
'''
```

```
Single quote can also be used instead double one
```

```
'''
```




Arithmetic operators

```
a = 10           # 10
```

```
a += 1           # 11
```

```
a -= 1           # 10
```

```
b = a+1          # 11
```

```
c = a-1          # 9
```

```
d = a * 2        # 20
```

```
e = a / 2        # 5
```

```
f = a % 3        # 1
```

```
g = a ** 2       # 100
```



Other operators

- Logical operators

- logical AND `a and b`
- logical OR `a or b`
- negation `not (a)`

- Arithmetic comparison

- Ordering `> >= < <=`
- Equality `==`
- Difference `!=`



Conditionals

```
temperature = 22

if temperature < 15:
    print("cold")
elif temperature >=16 and temperature <25:
    print("warm")
else:
    print("cold")
```

for loop

```
fruits = ['apple', 'banana', 'kiwi']  
for f in fruits:  
    print(f)
```



```
apple  
banana  
kiwi
```

```
primes = [2, 3, 5, 7]  
for n in primes:  
    print(n)
```



```
2  
3  
5  
7
```

```
misc = [1, '1', 'joe']  
for m in misc:  
    print(m, end="")  
    print(": ", end="")  
    print(type(m))
```



```
1: <class 'int'>  
1: <class 'str'>  
joe: <class 'str'>
```

```
for i in range(0,3):  
    print(f)
```



```
0  
1  
2
```

for loop with dictionaries

```
persons = {  
    'Andrea': 21,  
    'Fabio': 22,  
    'Simone': 31  
}  
for key, value in persons.items():  
    print("%s, %s" % (key, value))
```



```
Andrea, 21  
Fabio, 22  
Simone, 31
```

while loop

```
x = 0  
while x < 3:  
    print(x)  
    x += 1
```



```
0  
1  
2
```

Functions



Function definition

```
def print_welcome():  
    print("Welcome to our powerful program.")  
    print("Type X to exit or C to continue")
```

```
def adder(n1, n2):  
    result = n1 + n2  
    return(result)
```

Function use

```
print_welcome()
```

```
c = adder(12, 40)
```

Functions



Default value for parameters

```
def hello(message="Hello World"):  
    print(message)  
  
hello()                # Hello World  
hello("Ciao Mondo!")  # Ciao Mondo!
```

Positional parameters and keyword

```
def hello(how_many, message="Hello World"):  
    print(how_many*message)  
  
hello(1)                # Hello World  
hello(2, message="Ciao Mondo!")  
                        # Ciao Mondo!Ciao Mondo!
```

Indentation

```
/* C code */  
C = 0;  
if (A > 6) {  
    if (B < 3) {  
        C = A + B;  
    } else {  
        C = A - B;  
    }  
}
```

```
/* C code */  
C = 0;  
if (A > 6) {  
    if (B < 3) {  
        C = A + B;  
    }  
    else {  
        C = A - B;  
    }  
}}
```

INPUT:
A = 7 B = 2
A = 7 B = 4

A = 4 ...



OUTPUT:
C = 9
C = 3

C = 0

EQUAL!

INPUT:
A = 7 B = 2
A = 7 B = 4

A = 4 ...



OUTPUT:
C = 9
C = 3

C = 0

Indentation

```
# Python code
```

```
C = 0
```

```
if A > 6:
```

```
    if B < 3:
```

```
        C = A + B
```

```
    else:
```

```
        C = A - B
```

```
# Python code
```

```
C = 0
```

```
if A > 6:
```

```
    if B < 3:
```

```
        C = A + B
```

```
else:
```

```
    C = A - B
```

INPUT:

A = 7 B = 2

A = 7 B = 4

A = 4 ...



OUTPUT:

C = 9

C = 3

C = 0

DIFFERENT!

INPUT:

A = 7 B = 2

A = 7 B = 4

A = 4 ...



OUTPUT:

C = 9

C = 0

C = 3





DATA TYPES



Data types

Python is a dynamic language but it is also strongly typed.
The interpreter keeps track of all variable types

```
>>> my_number = 123
```

```
>>> type(my_number)
```

```
<class 'int'>
```

```
>>> my_string = "a string"
```

```
>>> type(my_string)
```

```
<class 'str'>
```

```
>>> print(my_string + my_number)
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: Can't convert 'int' object to str implicitly
```



Boolean

```
# simple boolean
```

```
is_python = True
```

```
# Everything can be converted to boolean
```

```
is_python = bool("yes sure!")
```

```
# some things are equivalent to False
```

```
these_are_false = False or 0 or "" or {} or [] or None
```

```
# others are True
```

```
these_are_true = True and 1 and 2 and "Some Text" and {'foo': 'bar'} and [1, 1, 2, 3, 5, 8]
```



Numbers

```
# Integers
```

```
year = 2016
```

```
year = int("2016")
```

```
# Floating point
```

```
pi = 3.14159265
```

```
pi = float("3.14159265")
```

```
# Fixed Point
```

```
from decimal import Decimal
```

```
price = Decimal("0.02")
```



Strings

```
name = "I'm a string"
```

```
me_too = 'I am also a string using "am" instead of "m" '
```

```
multiline = """And I am  
a multiline string that is  
splitted on more than one line"""
```

```
multiline2 = '''also with  
single quotes'''
```



String manipulation

```
fullname = "John" + " " + "Doe"           # John Doe
fullname += " is my name"                  # John Doe is my name
```

```
fullname = " ".join(["John", "Doe", "is my name"])
```

```
# this will give "Dec 31 1989"
my_string_date = '%s %d %d' % ('Dec', 31, 1989)
```

```
# this will give "John Doe is 27 years old"
my_label = '%(first)s %(last)s is %(age)d years old'
           % {'first': 'John', 'last': 'Doe', 'age': 27}
```



List

Ordered sequence of values

```
# initializing list
empty_list = []
my_list = [1,5,10]
my_heterogeneous_list = [1, "foo", "bar", True]
nested_list = [1, [1, 10]]
```

```
#accessing list elements
len(my_list)           # => 3
print(my_list[0])      # => 1
print(my_list[0:2])    # => [1,5]
print(my_list[1:])     # => [5,10]
```

```
# adding elements
mylist.append(42)
mylist.extend(['python','rulez', True])
```




- L.append(x)
 - L.insert(i, x)
 - L.remove(x)
 - L.pop(i)
- L.pop()
 - L.index(x)
 - L.count(x)
 - L.sort()
 - L.reverse()

Operation	Meaning
seq[i]	i-th element of the sequence
len(seq)	Length of the sequence
seq1 + seq2	Concatenate the two sequences
num*seq seq*num	Repeat seq num times
seq[start:end]	slice starting from start , and ending at end-1
e in seq	True if e is present in seq, False otherwise
e not in seq	True if e is not present in seq, False otherwise
for e in seq	Iterate over all elements in seq (e is bound to one element per iteration)



Sets

- An unordered collection with no duplicate elements
- Supports
 - eliminating duplicate entries
 - Set operations: union, intersection, difference, and symmetric difference.



Sets

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> fruits = set(basket)
>>> fruits
set(['orange', 'pear', 'apple', 'banana'])
>>> type(fruits)
<type 'set'>

>>> 'apple' in fruits
True
>>> 'mango' in fruits
False
```



Set Operations

```
>>> A=set('acads')
>>> B=set('institute')
>>> A
set(['a', 's', 'c', 'd'])
>>> B
set(['e', 'i', 'n', 's', 'u', 't'])
>>> A - B # Set difference
set(['a', 'c', 'd'])
>>> A | B # Set Union
set(['a', 'c', 'e', 'd', 'i', 'n', 's', 'u', 't'])
>>> A & B # Set intersection
set(['s'])
>>> A ^ B # Symmetric Difference
set(['a', 'd', 'c', 'e', 't', 'i', 'u', 'n'])
```



Dictionaries

- Unordered set of *key:value* pairs,
- Keys have to be unique and immutable
- Key:value pairs enclosed inside curly braces {...}
- Empty dictionary is created by writing {}
- Dictionaries are mutable
 - add new key:value pairs,
 - change the pairing
 - delete a key (and associated value)

Operation	Meaning
<code>len(d)</code>	Number of key:value pairs in d
<code>d.keys()</code>	List containing the keys in d
<code>d.values()</code>	List containing the values in d
<code>k in d</code>	True if key k is in d
<code>d[k]</code>	Value associated with key k in d
<code>d.get(k, v)</code>	If k is present in d, then d[k] else v
<code>d[k] = v</code>	Map the value v to key k in d (replace d[k] if present)
<code>del d[k]</code>	Remove key k (and associated value) from d
<code>for k in d</code>	Iterate over the keys in d



Operations on Dictionaries

```
>>> capital = {'India':'New Delhi', 'USA':'Washington DC', 'France':'Paris', 'Sri Lanka':'Colombo'}
>>> capital['India'] # Get an existing value
'New Delhi'
>>> capital['UK'] # Exception thrown for missing key

Traceback (most recent call last):
  File "<pyshell#130>", line 1, in <module>
    capital['UK'] # Exception thrown for missing key
KeyError: 'UK'
>>> capital.get('UK', 'Unknown') # Use of default
value with get
'Unknown'
>>> capital['UK']='London' # Add a new key:val pair
>>> capital['UK'] # Now it works
'London'
```




Operations on Dictionaries

```
>>> capital.keys()
['Sri Lanka', 'India', 'UK', 'USA', 'France']
>>> capital.values()
['Colombo', 'New Delhi', 'London', 'Washington DC',
'Paris']
>>> len(capital)
5
>>> 'USA' in capital
True
>>> 'Russia' in capital
False
>>> del capital['USA']
>>> capital
{'Sri Lanka': 'Colombo', 'India': 'New Delhi', 'UK':
'London', 'France': 'Paris'}
```



Dictionary

```
person = {  
    'name': 'John',  
    'surname': "Doe"  
}  
  
person['age'] = 25                # add field  
print(person['name'])            # => John  
  
>>> person.keys()  
dict_keys(['age', 'name', 'surname'])  
  
>>> person.values()  
dict_values([25, 'John', 'Doe'])  
  
>>> person.items()  
dict_items([('age', 25), ('name', 'John'), ('surname', 'Doe')])
```



Class example

```
lion  
cat
```

```
# class definition  
class Animal(object):  
    # constructor of the class  
    def __init__(self, n):  
        self.name = n  
  
    # a method  
    def get_name(self):  
        return(self.name)  
  
### instances of class  
lion = Animal('lion')  
cat = Animal('cat')  
  
animals = [lion, cat]  
for a in animals:  
    print(a.get_name())
```



Packages and modules

Modules

- python files with .py extension that implements functions or classes
- imported into the code using `import` command
- python provides a set of built-in modules

Packages

- namespaces which contain multiple package or modules
- implemented as a directory containing modules or other package
- the directory MUST contain a file called `__init__.py` that can be empty



Packages and modules

```
# Imports datetime module into current namespace
import datetime
today = datetime.date.today()
yesterday = today - datetime.timedelta(days=1)
print(today)
print(yesterday)
```

```
# imports datetime and add date and timedelta
# into current namespace
from datetime import date, timedelta
today = date.today()
yesterday = today - timedelta(days=1)
print(today)
print(yesterday)
```



Frameworks

- Stand alone applications
 - Console
 - Tkinter (Tcl,Tk)
 - PyQT
 - wxPython
- Web application
 - Flask microframework
 - Django
- Numerical analysis
 - NumPy
 - SciPy



Extensions

PIP is the recommended tool to install python packages

```
pip install django  
pip install git+git://github.com/django/django.git#egg=django
```

Virtual environments

- a program that separates environments in order to isolate dependencies for different projects.



Domande?

masciadri.andrea@gmail.com

Ora provate voi..

Esercizi



Esercizio 1

- Calcolare il volume di un cilindro conoscendo raggio e altezza
- Vengono forniti da tastiera i risultati di un referendum:
 - Numero iscritti a votare
 - Numero votanti
 - Numero si
 - Numero no

Calcolare la percentuale di votanti sul totale degli iscritti, le percentuali dei si e dei no rispetto al numero dei votanti

Esercizio 2

- Scrivere un programma che legge in ingresso un carattere e stampa se corrisponde a una cifra o a una lettera alfabetica o a un altro carattere, nel caso sia una lettera segnalare il caso particolare che sia una vocale o una consonante e il caso particolare che sia minuscola. Es:
 - 3 cifra
 - D lettera consonante
 - e lettera vocale minuscola
 - d lettera consonante minuscola
 - E lettera vocale
 - < altro carattere



Esercizi simili

- Date le misure a e b di un rettangolo potenziale (a e b possono essere nulle o negative) si vuole sapere si tratta di un quadrato, di un rettangolo, di un segmento, di un punto o di un caso impossibile
- Dati tre numeri interi calcolare il massimo e stamparlo



Esercizi simili

- Data una sequenza di numeri naturali che finisce con 0 trovare il massimo dei numeri inseriti
- Visualizzare a video la parola ciao un numero di volte scelto dall'utente
- Calcolare il prodotto di una sequenza di numeri che termina con 0
- Stampare il valore della potenza di base m ed esponente $n > 0$



Esercizi simili

- data una sequenza di n punti nel piano con n scelto dall'utente, stampare la distanza massima dall'origine degli assi dei punti introdotti
- Modificare il programma precedente stampando le coordinate del punto di distanza massima
- Modificare il programma precedente stampando la distanza media



References

- Masciadri Andrea, Introduzione a Python
- Algoritmi, Politecnico di Milano