Comandi

* Set\_Mode = utilizzato per definire le dimensioni dello schermo di gioco in pixel
* pygame.display.set\_caption(‘nome a scelta’): Evita di digitare sempre “pygame” rpima di ogni comando nel codice
* pygame.event.get(): serve a creare eventi, tipo associare una determinata azione ad un pulsante;
  + for event in pygame.event.get():  
     if event.type == pygame.KEYDOWN and event.key == pygame.K\_SPACE:

inizializza()

ricomincia = True

if event.type == pygame.QUIT:

pygame.quit()

* nome variabile: pygame.image.load(): serve per caricare una determinata immagine e attribuirgli un nome specifico da utilizzare all’interno del programma come variabile  
  Attenzione: le immagini devono essere nella stessa cartella del programma o in una suo sottocartella; in questo ultimo caso indicare il percorso con i nomi delle sole sottocartelle tipo:  
  auto=pygame.image.load('images/auto.png')
* Utilizziamo l’oggetto **Clock**, un oggetto utilizzato per misurare i millesimi di secondo.  
  Specifichiamo l’FPS, ovvero il numero di frame in un secondo, che verrà passato come argomento al metodo **tick**().  
  Il metodo **tick**lo chiameremo per ultimo e si occupa di arrestare il movimento per il tempo necessario ad ottenere la temporizzazione.
* pygame.transform.flip()funzione. Questa funzione ha tre parametri: l'oggetto Surface con l'immagine da capovolgere, un valore booleano per eseguire un capovolgimento orizzontale e un valore booleano per eseguire un capovolgimento verticale. Passando True per il secondo parametro e False per il terzo parametro, l'oggetto Surface che ritorna ha l'immagine dello scoiattolo rivolta verso destra:  
  grattacielo\_su = pygame.transform.flip(grattacielo\_giu, False, True)
* Classe: E' una caratteristica della programmazione orientata agli oggetti (OOP). L'uso delle classi **evita la ripetizione delle stesse informazioni su più oggetti** ed è più efficiente rispetto alla vecchia programmazione procedurale.  
  Creo la classe Persona:  
  class Persona():  
  '''Classe che identifica le persone.'''  
  def \_\_init\_\_(self, Nome, Cognome):  
   self.Nome = Nome  
   self.Cognome = Cognome  
  def \_\_str\_\_(self):  
   return "Nome: " + str(self.Nome) + "\nCognome: " + str(self.Cognome)
  + e' composta da due metodi: \_\_init\_\_ definisce le proprietà degli oggetti (self) della classe. In questo esempio ogni oggetto della classe Persona() ha due proprietà: Nome Cognome
  + \_\_str\_\_ ritorna una rappresentazione sotto forma di stringa dell'oggetto ass
  + Come associare un oggetto alla classe:  
    Per associare un oggetto alla classe digito: eric = Persona("Eric", "Idle") Il primo argomento ("Eric") è associato alla prima proprietà della classe Persona ossia Nome. Il secondo argomento ("Idle") è invece associato alla seconda proprietà ossia Cognome. A questo punto, visualizzo il contenuto della variabile eric con l'istruzione print:  
    print(eric)
* Metodo: nel linguaggio Python un metodo è una funzione associata a un oggetto e a una classe. E' uno degli aspetti fondamentali della programmazione a oggetti nel linguaggio python che unisce dati e programmi dentro un'unica entità ( oggetto ) anziché tenerli distinti.  
  **Qual è la differenza tra metodo e attributo?** Un metodo è una funzione associata all'oggetto mentre un attributo è un dato che caratterizza l'oggetto.  
  I metodi speciali più frequenti in una classe sono i seguenti:
  + \_\_init\_\_ E' il metodo costruttore. Viene eseguito automaticamente quando si richiama un oggetto appartenente alla classe.
  + \_\_str\_\_ Questo metodo converte un oggetto in una stringa.
  + \_\_del\_\_ E' il metodo distruttore. Elimina un oggetto.
* .get\_width(): scritto dopo il nome di un oggetto (es. un immgine png) mi restituisce la sua larghezza - es. uccello.get\_width()
* .get\_height(): scritto dopo il nome di un oggetto (es. un immgine png) mi restituisce la sua larghezza - es. uccello.get\_height()