EVU Python LESSON III

Dagens program

Velkomst

- Dagens program

Recap

- lister
- dicts
- functioner
- pygame

OOP

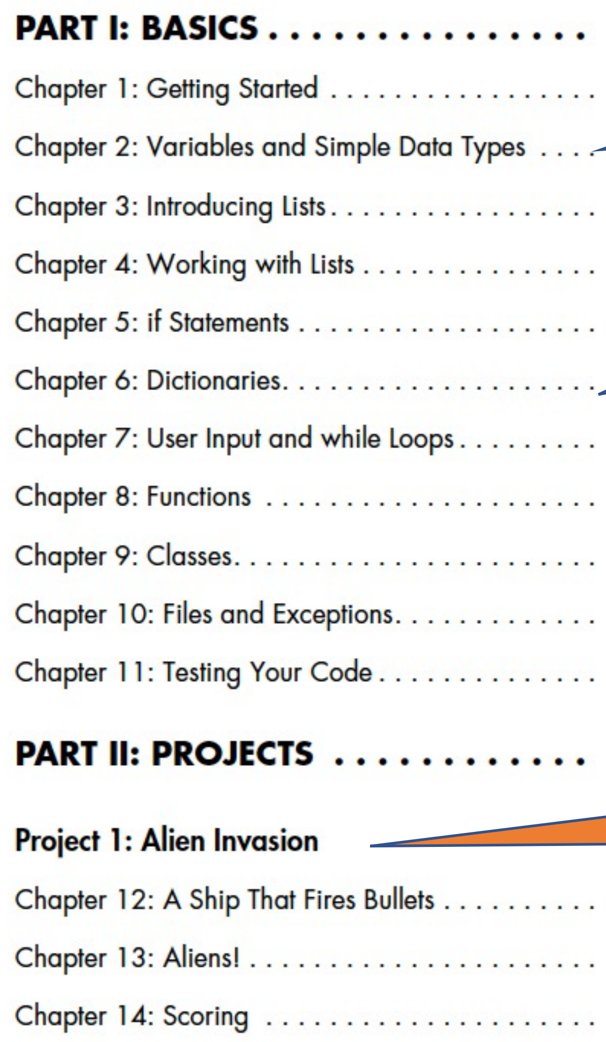
- basic
- OOP pygame

Pygame

- Non OOP
- OOP

Velkomst

kursets forløb



27/1

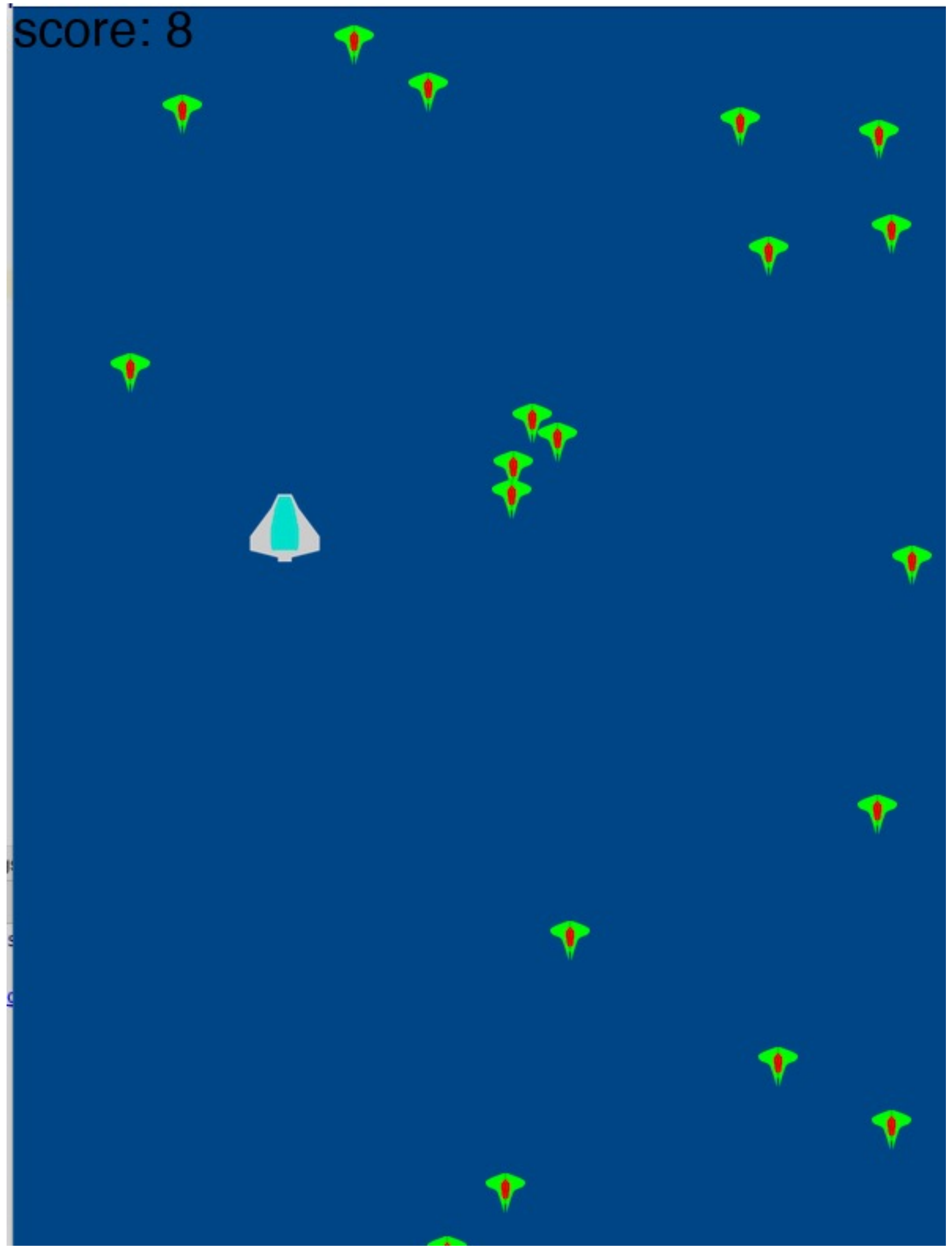
10/2

24/2

10/3

Teaser

Final pygame

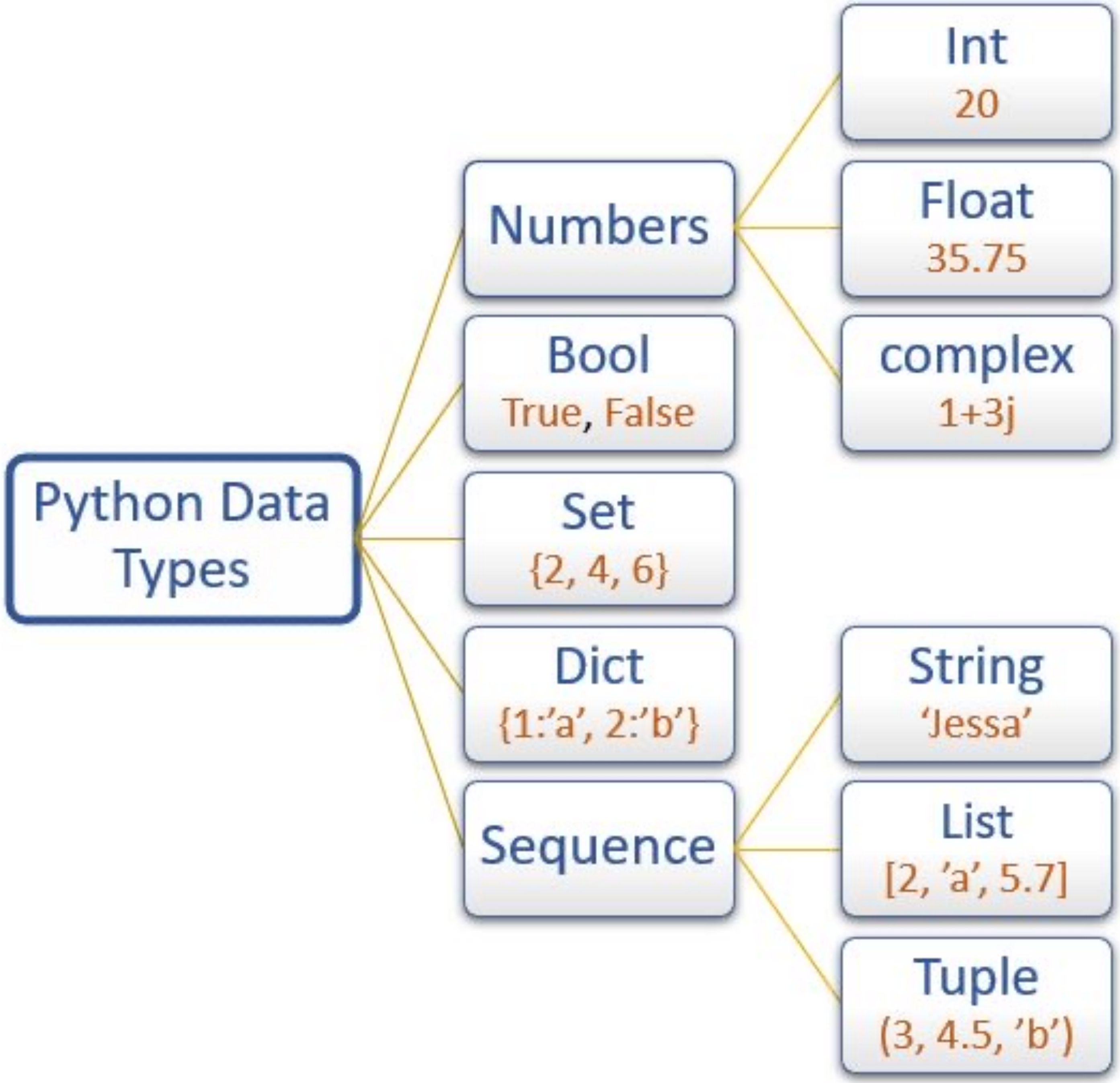


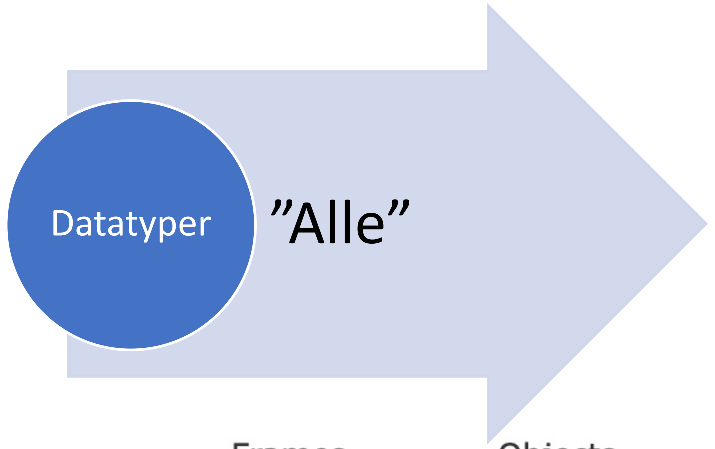
Datatyper

Overview

A Python variable is a reserved memory location to store values.

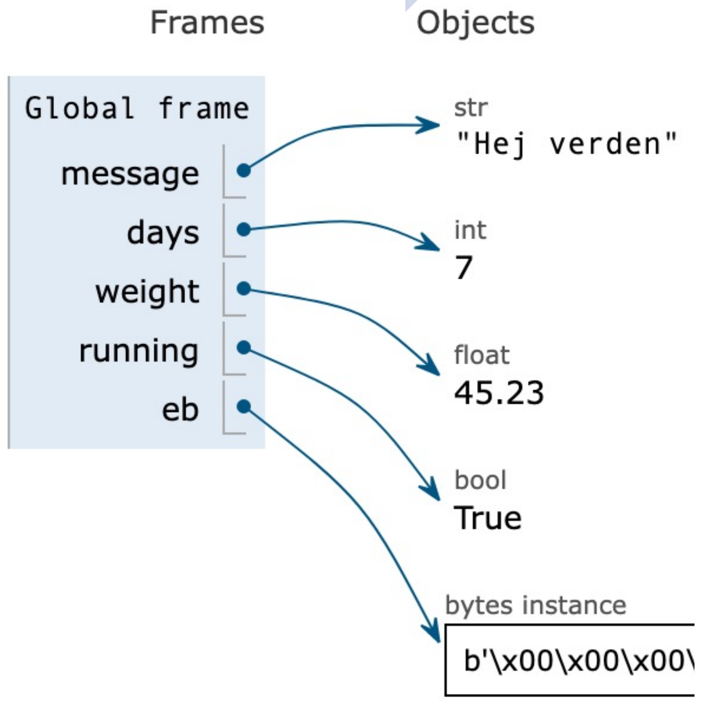
Every value in Python has a datatype.

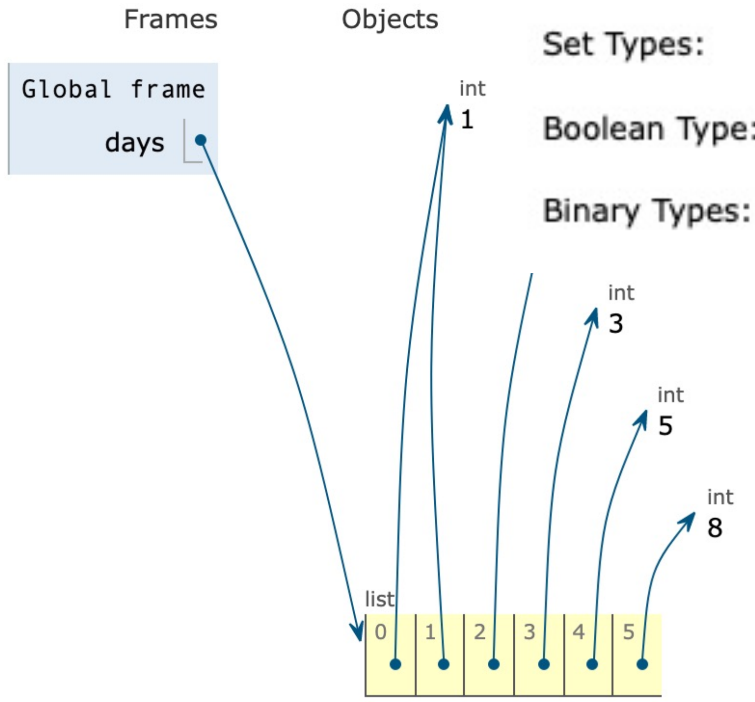


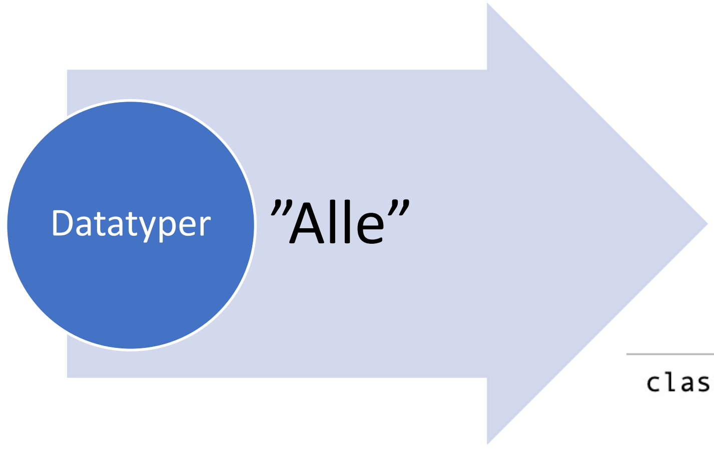


A Python variable is a reserved memory location to store values.
Every value in Python has a datatype.

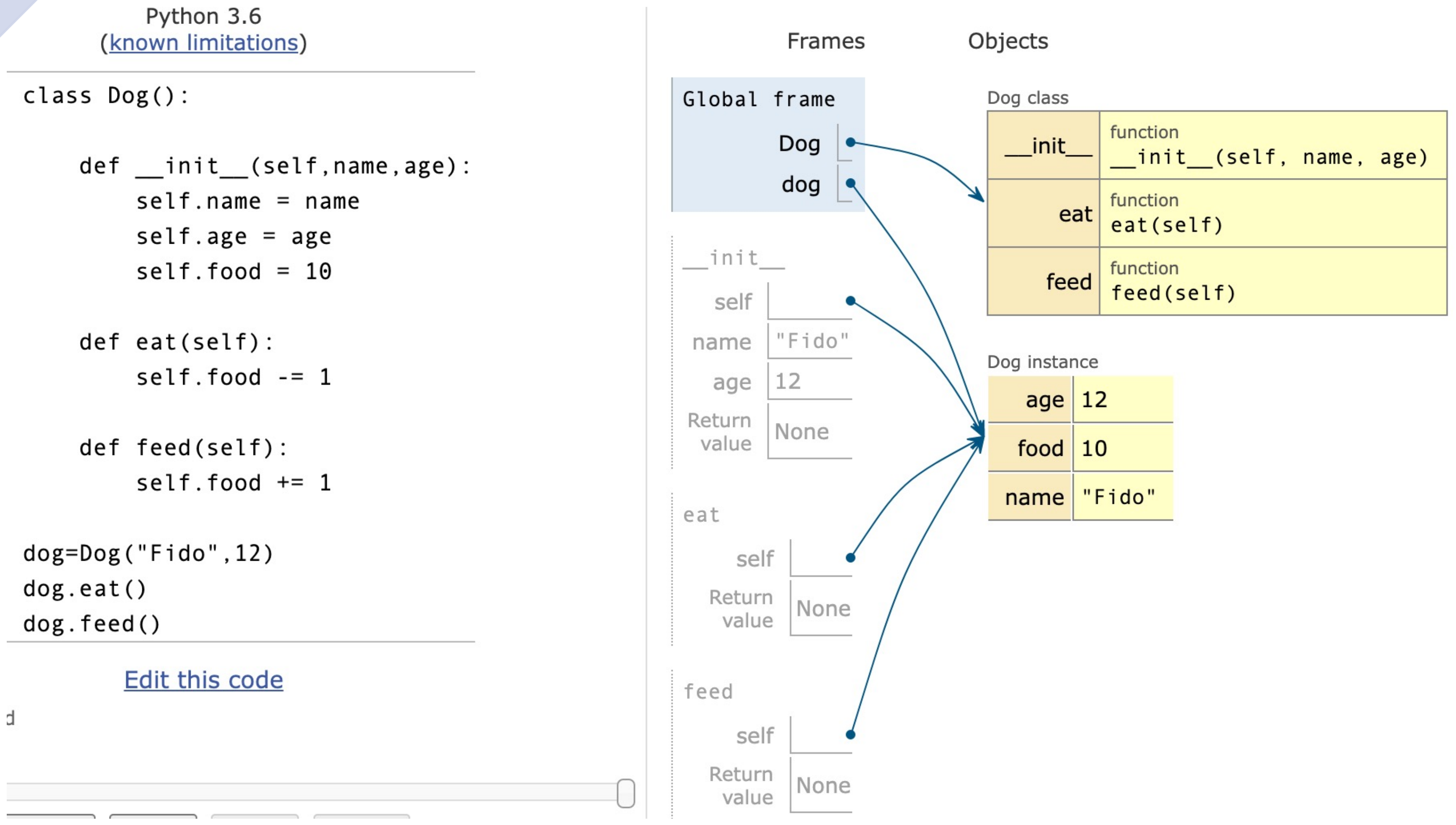
Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`







A Python variable is a reserved memory location to store values.
Every value in Python has a datatype.



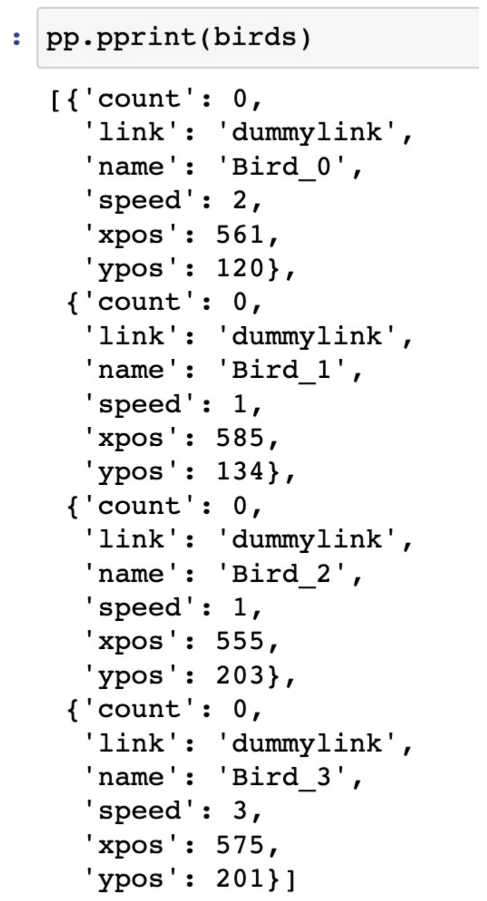
```
class Dog():

    def __init__(self,name,age):
        self.name = name
        self.age = age
        self.food = 10

    def eat(self):
        self.food -= 1

    def feed(self):
        self.food += 1

dog=Dog("Fido",12)
dog.eat()
dog.feed()
```

Pause

Datatyper

Dicts – hvad?

- A dictionary in Python is a collection of key-value pairs.
- Each key is connected to a value.
- You can use a key to access the value associated with that key.
- A key’s value can be a number, a string, a list, a dictionary or any object



```
: pp.pprint(birds)

[{'count': 0,
  'link': 'dummylink',
  'name': 'Bird_0',
  'speed': 2,
  'xpos': 561,
  'ypos': 120},
 {'count': 0,
  'link': 'dummylink',
  'name': 'Bird_1',
  'speed': 1,
  'xpos': 585,
  'ypos': 134},
 {'count': 0,
  'link': 'dummylink',
  'name': 'Bird_2',
  'speed': 1,
  'xpos': 555,
  'ypos': 203},
 {'count': 0,
  'link': 'dummylink',
  'name': 'Bird_3',
  'speed': 3,
  'xpos': 575,
  'ypos': 201}]
```

Dicts

Øvelse

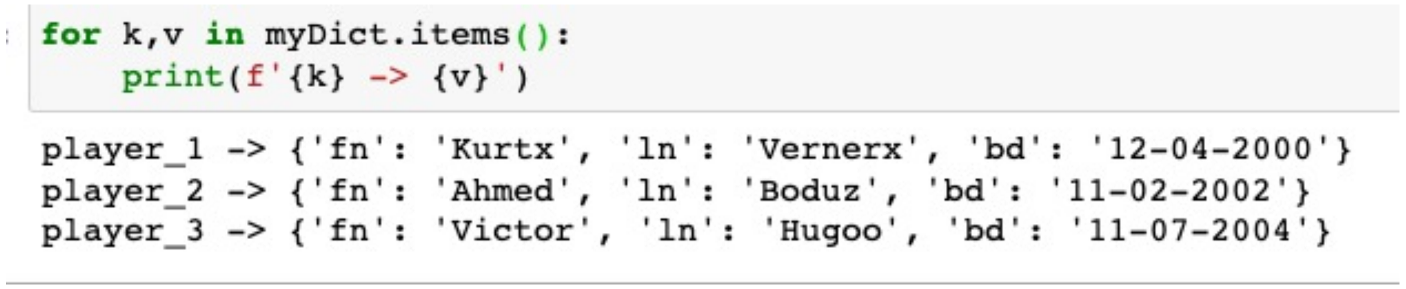
Dicts

Øvelse

Find Målet

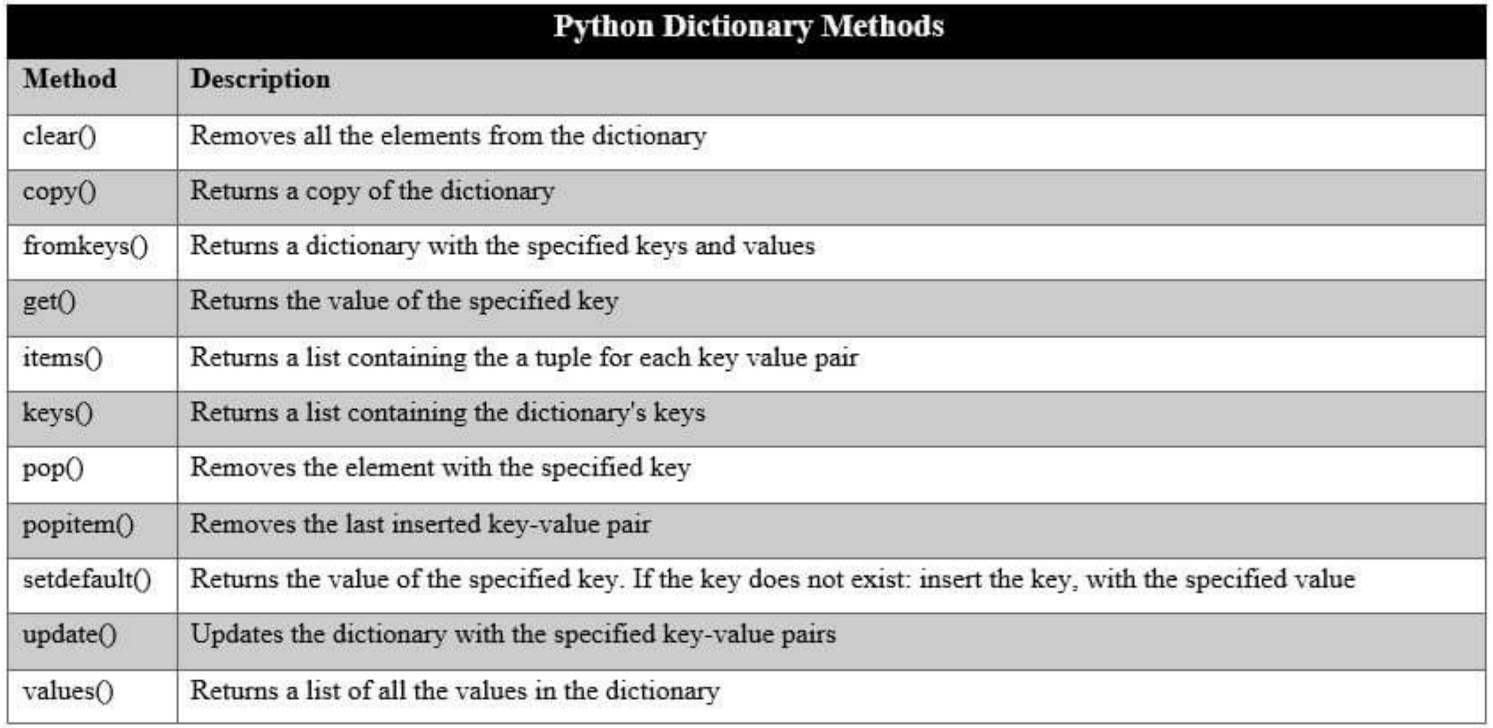
Datatyper

Dicts in action



```
for k,v in myDict.items():
    print(f'{k} -> {v}')
```

```
player_1 -> {'fn': 'Kurtx', 'ln': 'Vernerx', 'bd': '12-04-2000'}
player_2 -> {'fn': 'Ahmed', 'ln': 'Boduz', 'bd': '11-02-2002'}
player_3 -> {'fn': 'Victor', 'ln': 'Hugoo', 'bd': '11-07-2004'}
```



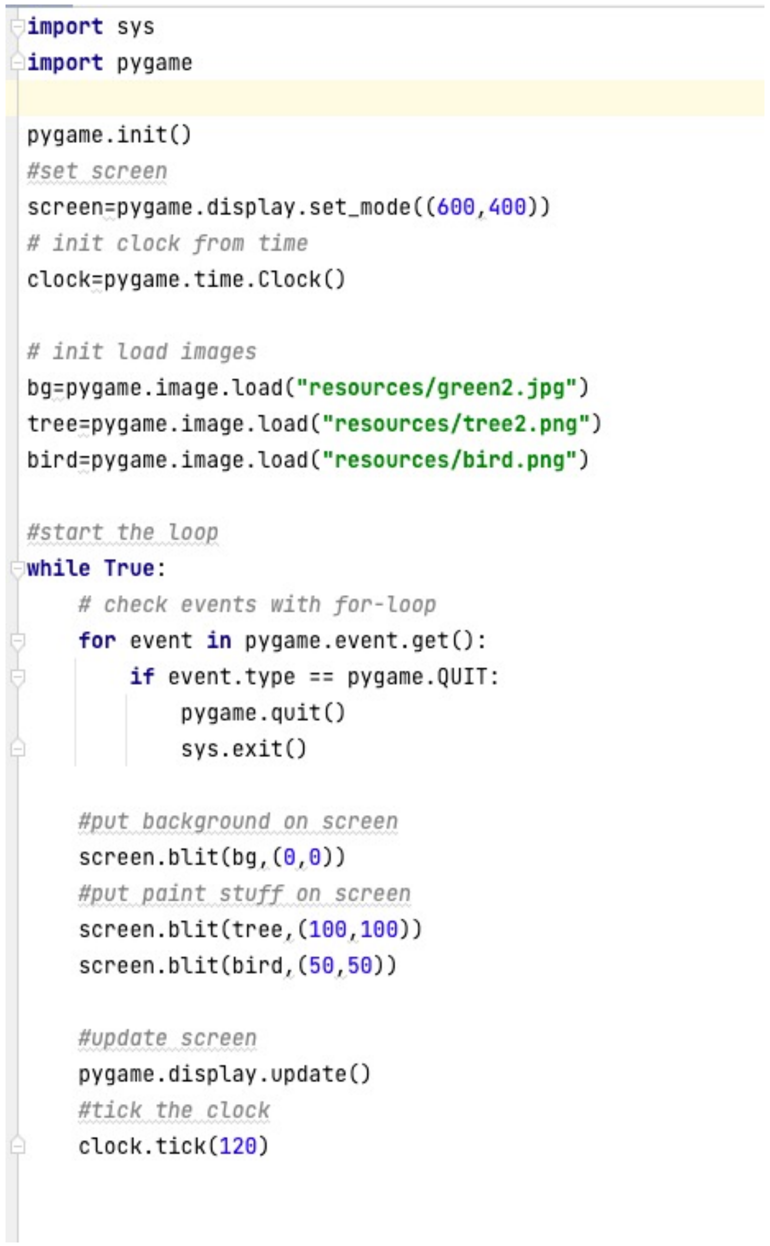
Python Dictionary Methods

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and values
get()	Returns the value of the specified key
items()	Returns a list containing the a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Pause

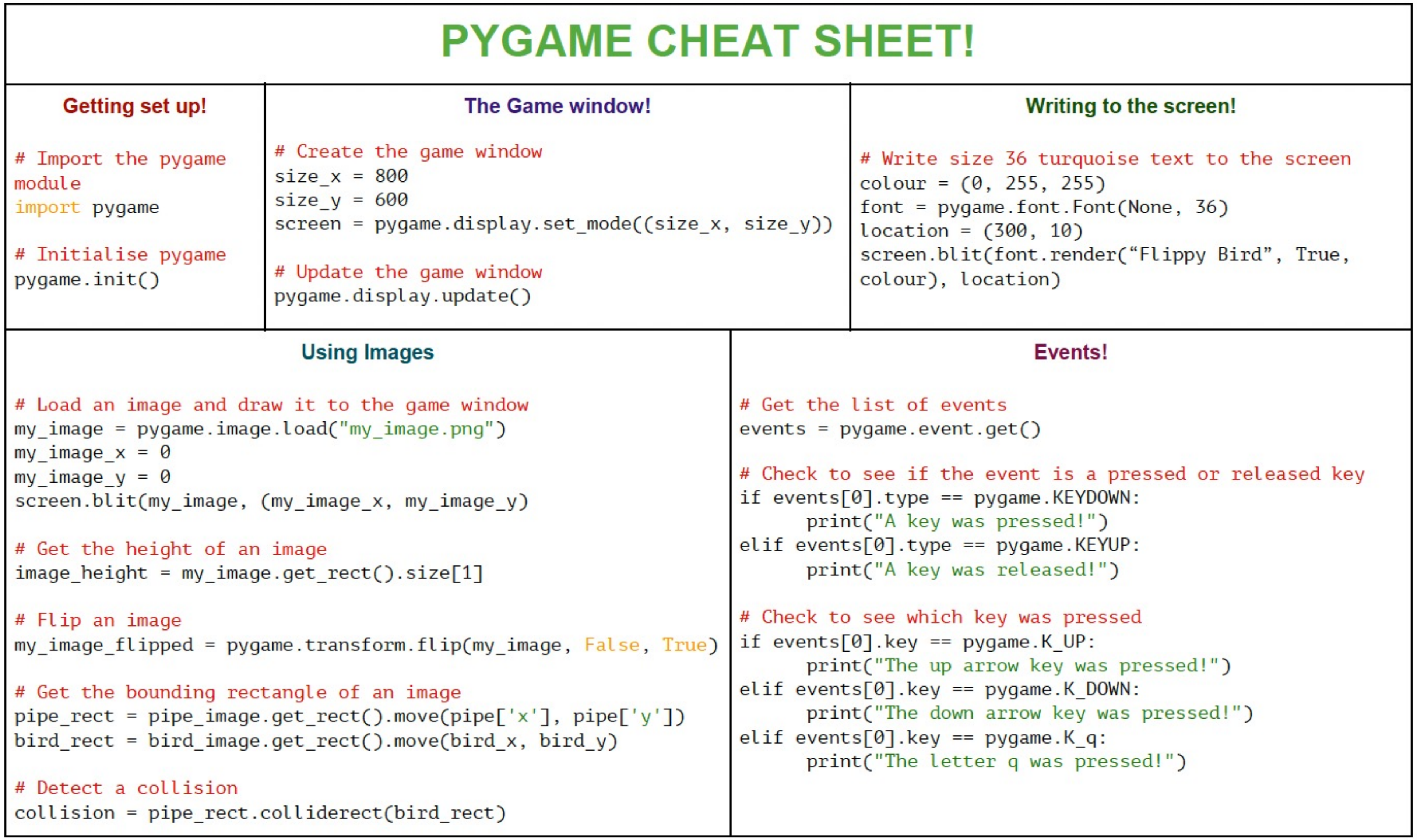
Pygame

Intro



```
import sys
import pygame

pygame.init()
#set screen
screen=pygame.display.set_mode((600,400))
# init clock from time
clock=pygame.time.Clock()

# init load images
bg=pygame.image.load("resources/green2.jpg")
tree=pygame.image.load("resources/tree2.png")
bird=pygame.image.load("resources/bird.png")

#start the loop
while True:
    # check events with for-loop
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

    #put background on screen
    screen.blit(bg,(0,0))
    #put paint stuff on screen
    screen.blit(tree,(100,100))
    screen.blit(bird,(50,50))

    #update screen
    pygame.display.update()
    #tick the clock
    clock.tick(120)
```

Pygame



PYGAME CHEAT SHEET!

Getting set up!

```
# Import the pygame
module
import pygame

# Initialise pygame
pygame.init()
```

The Game window!

```
# Create the game window
size_x = 800
size_y = 600
screen = pygame.display.set_mode((size_x, size_y))

# Update the game window
pygame.display.update()
```

Writing to the screen!

```
# Write size 36 turquoise text to the screen
colour = (0, 255, 255)
font = pygame.font.Font(None, 36)
location = (300, 10)
screen.blit(font.render("Flippy Bird", True,
colour), location)
```

Using Images

```
# Load an image and draw it to the game window
my_image = pygame.image.load("my_image.png")
my_image_x = 0
my_image_y = 0
screen.blit(my_image, (my_image_x, my_image_y))

# Get the height of an image
image_height = my_image.get_rect().size[1]

# Flip an image
my_image_flipped = pygame.transform.flip(my_image, False, True)

# Get the bounding rectangle of an image
pipe_rect = pipe_image.get_rect().move(pipe['x'], pipe['y'])
bird_rect = bird_image.get_rect().move(bird_x, bird_y)

# Detect a collision
collision = pipe_rect.colliderect(bird_rect)
```

Events!

```
# Get the list of events
events = pygame.event.get()

# Check to see if the event is a pressed or released key
if events[0].type == pygame.KEYDOWN:
      print("A key was pressed!")
elif events[0].type == pygame.KEYUP:
      print("A key was released!")

# Check to see which key was pressed
if events[0].key == pygame.K_UP:
      print("The up arrow key was pressed!")
elif events[0].key == pygame.K_DOWN:
      print("The down arrow key was pressed!")
elif events[0].key == pygame.K_q:
      print("The letter q was pressed!")
```

Pygame

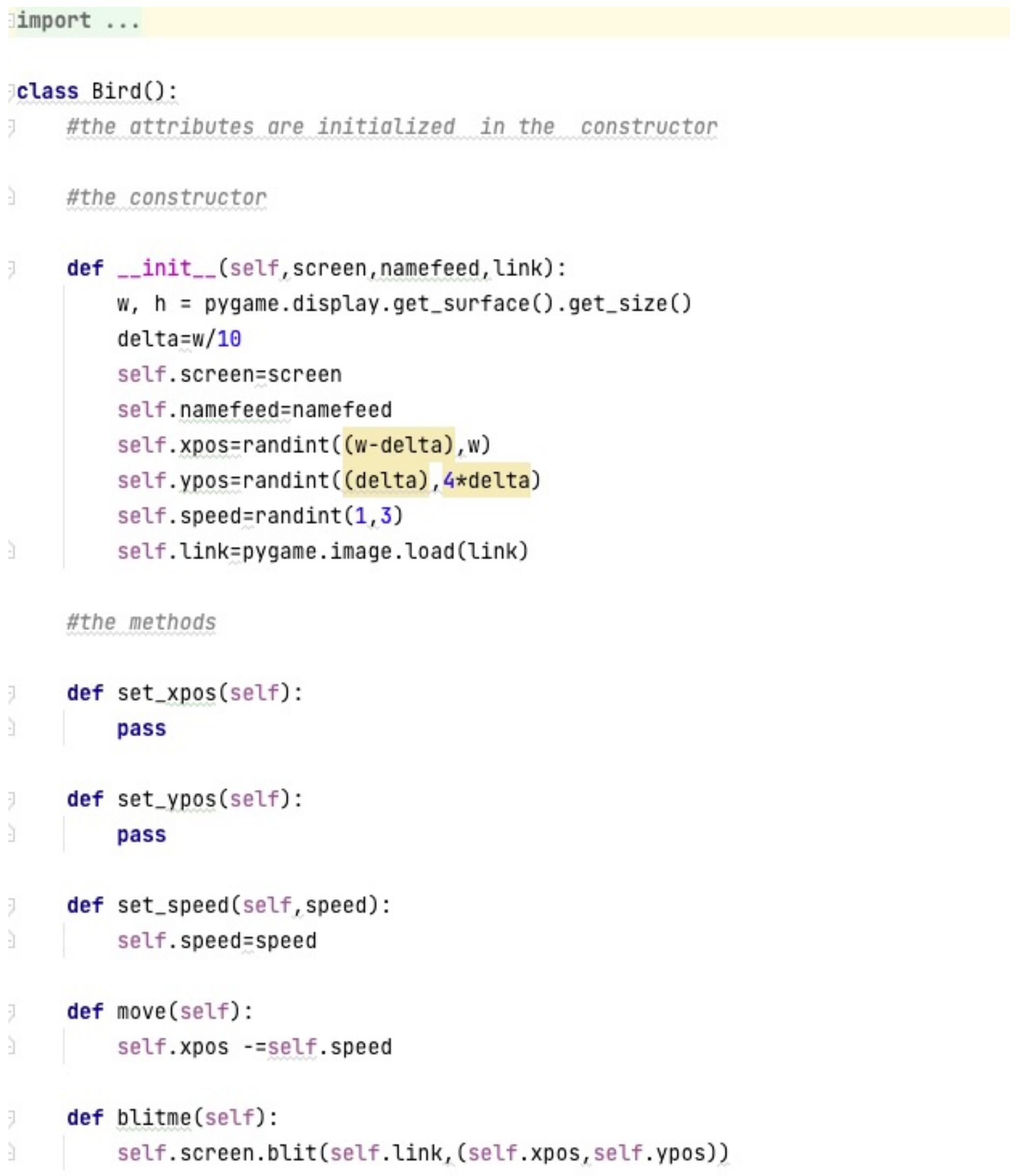
Øvelser

- Find din egen baggrund
- Find target
- Find sigtekorn
- Lav basic setup

Pause

OOP

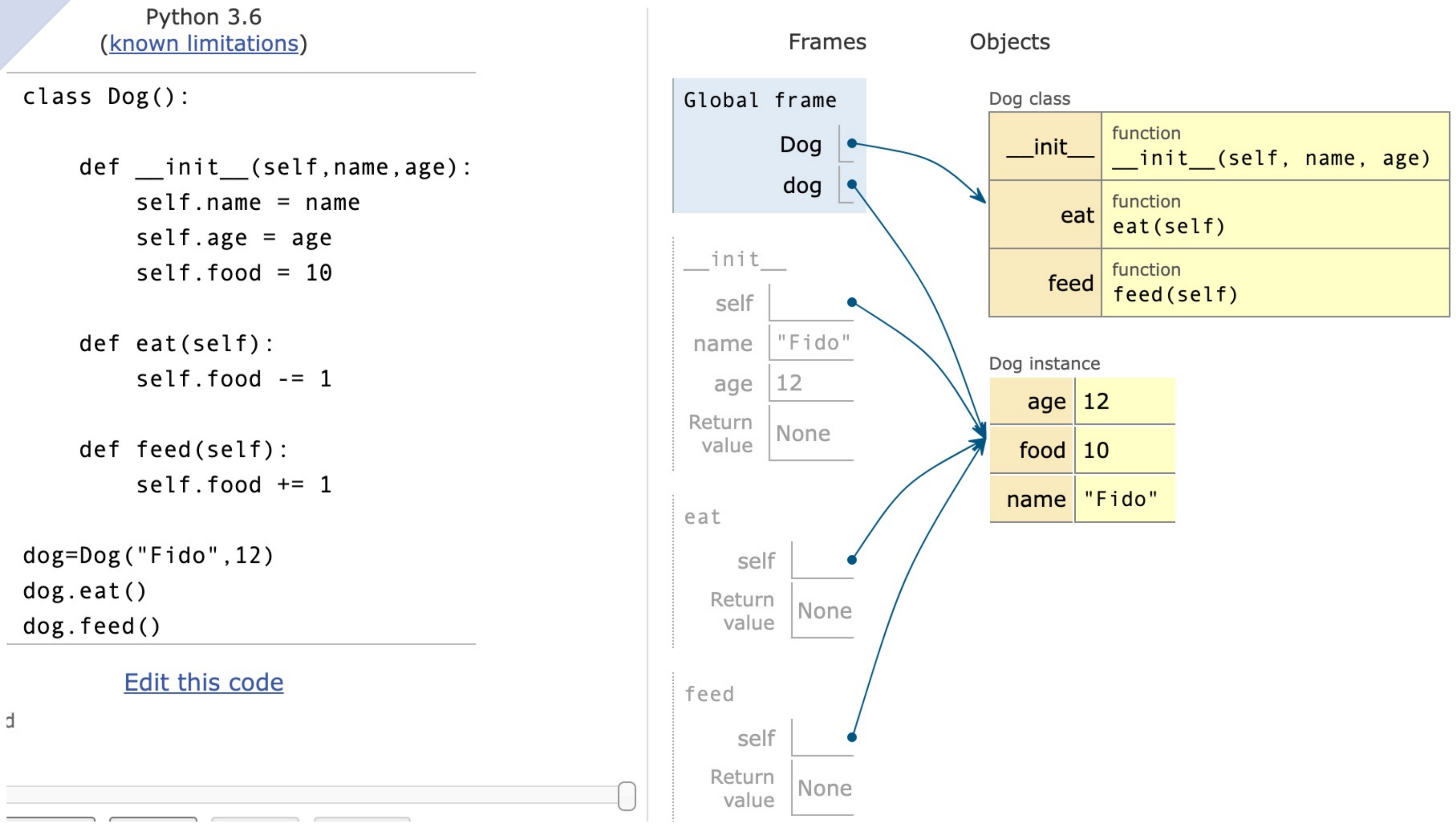
Basic intro



```
import ...

class Bird():
    #the attributes are initialized  in the  constructor

    #the constructor

    def __init__(self,screen,namefeed,link):
        w, h = pygame.display.get_surface().get_size()
        delta=w/10
        self.screen=screen
        self.namefeed=namefeed
        self.xpos=randint((w-delta),w)
        self.ypos=randint((delta),4*delta)
        self.speed=randint(1,3)
        self.link=pygame.image.load(link)

    #the methods

    def set_xpos(self):
        pass

    def set_ypos(self):
        pass

    def set_speed(self,speed):
        self.speed=speed

    def move(self):
        self.xpos -=self.speed

    def blitme(self):
        self.screen.blit(self.link,(self.xpos,self.ypos))
```

OOP

Basic intro



Python 3.6
(known limitations)

```
class Dog():

    def __init__(self,name,age):
        self.name = name
        self.age = age
        self.food = 10

    def eat(self):
        self.food -= 1

    def feed(self):
        self.food += 1

dog=Dog("Fido",12)
dog.eat()
dog.feed()
```

Edit this code

OOP

Bogen

Object-oriented programming is one of the most effective approaches to writing software. In object-oriented programming you write classes that represent real-world things and situations, and you create objects based on these classes.

- Creating and Using a Class
    - The __init__() Method
    - Making an Instance from a Class
    - Accessing Attributes
    - Calling Methods
    - Creating Multiple Instances
    - Setting a Default Value for an Attribute
    - Modifying Attribute Values
        - Directly
        - Via a method (validation)
        - Incrementing an Attribute’s Value Through a Method

OOP

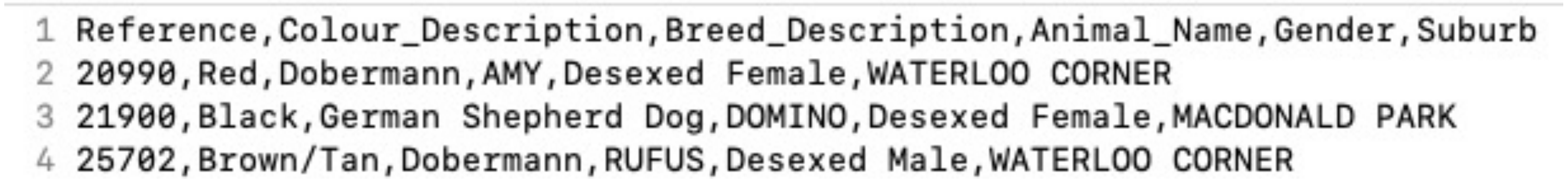
Øvelse 1

Hunden skal spise. Dvs den har en mad-attribut og en spise-metode. Men den kan kun spise hvis den har mad. Man skal også kunne fodre den. Den skal også have et hunde-id, køn, farve og race.

Hundene bor i en hundegård. Gården har en liste over hunde. Man kan sætte en hund ind og man kan fjerne den fra gården. Man kan også få en liste over hundene og hvor meget mad de samlet har tilbage.

OOP Øvelse 2

Hundene ligger nu i en regnskabsbog som er eksporteret til csv-fil-formatet. Indlæs filen og lav hunde vha filens indhold.



```
Reference,Colour_Description,Breed_Description,Animal_Name,Gender,Suburb
20990,Red,Dobermann,AMY,Desexed Female,WATERLOO CORNER
21900,Black,German Shepherd Dog,DOMINO,Desexed Female,MACDONALD PARK
25702,Brown/Tan,Dobermann,RUFUS,Desexed Male,WATERLOO CORNER
```

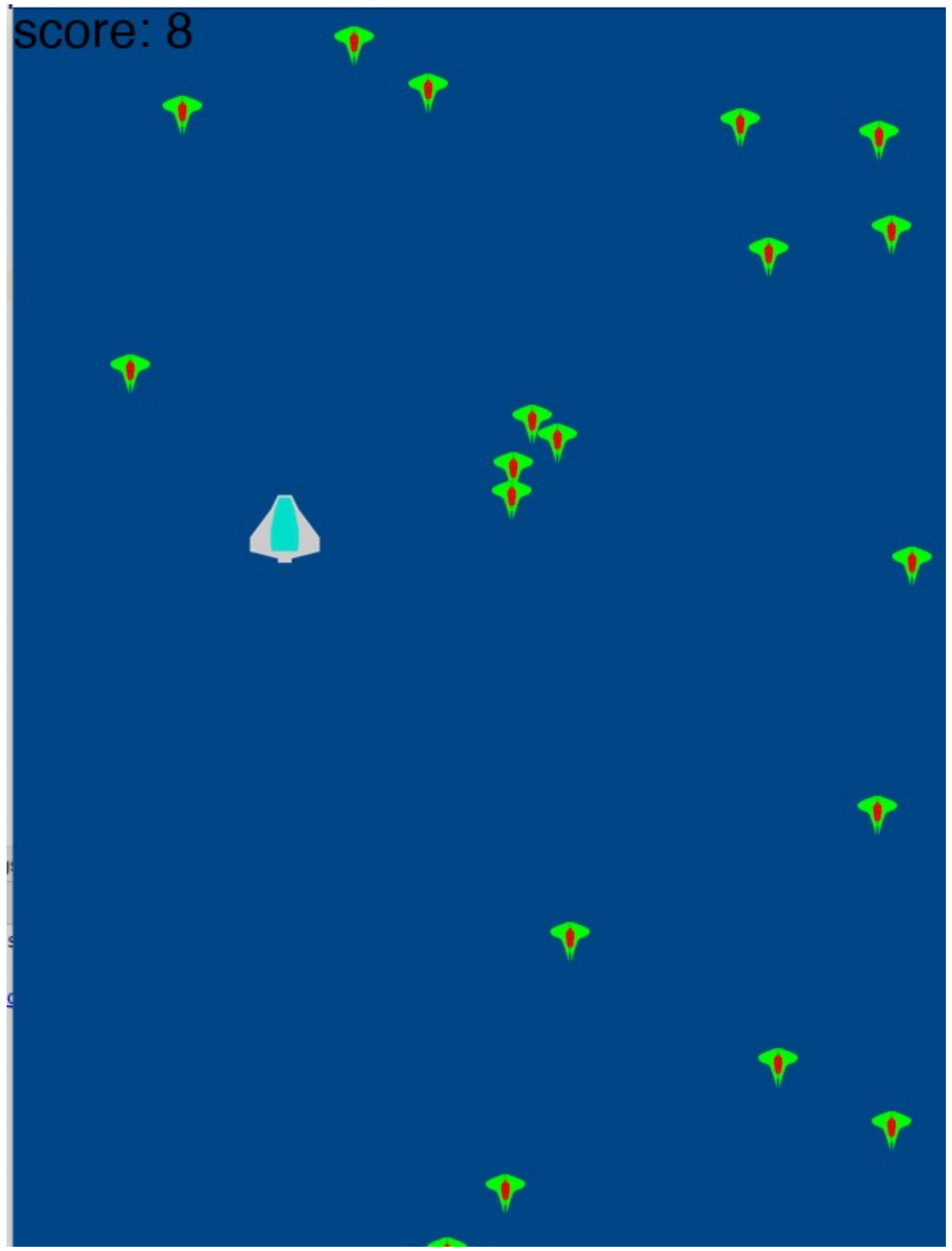
Pygame

Bogen

- Starting the Game Project
    - Creating a Pygame Window and Responding to User Input
    - Setting the Background Color
    - Creating a Settings Class
- Adding the Ship Image
    - Creating the Ship Class
    - Drawing the Ship to the Screen
- Refactoring: the game_functions Module
    - The check_events() Function
    - The update_screen() Function
- Piloting the Ship
    - Responding to a Keypress
    - Allowing Continuous Movement
    - Moving Both Left and Right
    - Adjusting the Ship’s Speed
    - Limiting the Ship’s Range
- Refactoring check_events()
- A Quick Recap
    - alien_invasion.py
    - settings.py
    - game_functions.py
    - ship.py

Pygame

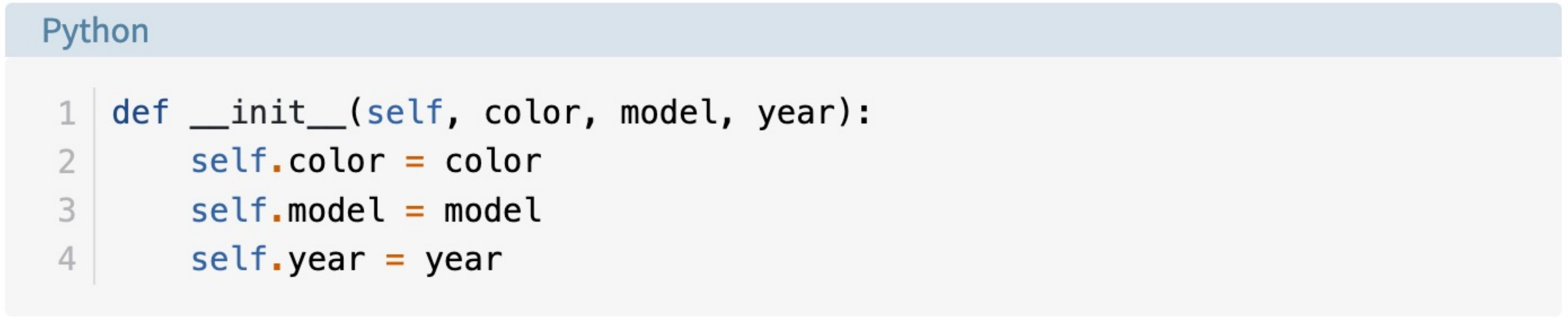
Final pygame



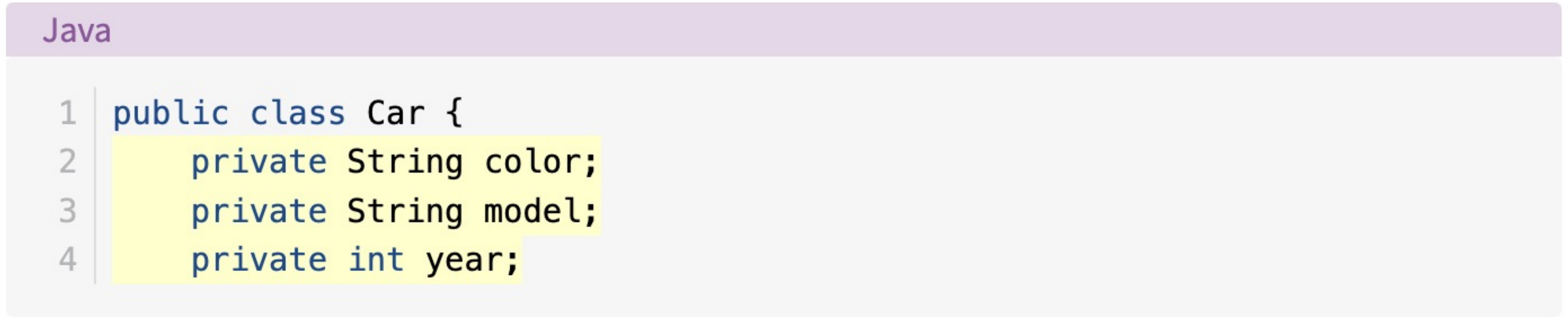
OOP

Java vs Python

One of the most significant differences between Python vs Java is how they define and manage class and object attributes.


Python

```python
def __init__(self, color, model, year):
    self.color = color
    self.model = model
    self.year = year
```


Java

```java
public class Car {
    private String color;
    private String model;
    private int year;
```