Object Oriented Programming cheatsheet

Classes

`class` keyword to create a object or type `__init__(self)` method used to create the constructor `self` reference to current instance of class, passed in as first argument of a method

```
# create a new object called Person
class Person:
  # Constructor
  def __init__(self, name, age):
    # Properties
    self.name = name
    self.age = age

  # Method
  def increment_age(self):
    self.age += 1

  def introduce_yourself(self):
    print(f"Hello my name is {self.name}, I am {self.age} years old.")

john = Person('John', 10) # is an object of type `Person`

# properties(props) and methods using dot notation
john.age # 10
john.increment_age()
john.age # 11
```

Type Hinting

- The type for each argument is defined after the colon `a: int`.
- The return type of the function is shown by the arrow `-> int`.

```
from typing import List, Dict  # Import the List and Dictionary types

# Arguments a and b are of type int
def add_numbers(a: int, b: int) -> int: # This function return an int
    return a + b

# Arguments a and b are of type string
def add_strings(a: str, b: str) -> str: # This function returns a str
    return a + b

add_numbers(1, 2)
add_strings("hello", " world")
```

```
def print_cars(cars: List[Dict[str, str]]):  # The cars argument is a list
of dictionaries
    for car in cars:
        print(f"Brand: {car['brand']}, Colour: {car['colour']}")

# This cars variable is type List[Dict[str, str]]
cars = [{"brand": "BMW", "colour": "Blue"},
        {"brand": "Volvo", "colour": "Red"}]

print_cars(cars)
```

Type hinting with objects

```
class Person():
    name = "Jane"
    age = 26

# This function takes an argument person which is of type Person
def greet_person(person: Person):
    print(f"Hello {person.name}")


jane = Person()

greet_person(jane)

def greet_people(people: List[Person]):
    for person in people:
        greet_person(person)
```

Inheritance

- The process by which one class takes on the attributes/methods or another

```
class Person: # parent class
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def increment_age(self):
    self.age += 1

  def welcome(self):
    print(f'Hi {name}. You are {age}.')

john = Person('John', 10)
john.welcome()

class Student(Person): # this class is a child of Person
```

```
    def __init__(self, name, age, course):
        super().__init__(name, age) # initialise the parent object
        self.course = course # set attribute only associate with this
child class

    def welcome(self): # overrides the welcome function of the parent
class
        print(f'Hi {name}, welcome to the {course}')

class Instructor: # this class is a child of Person
    def __init__(self, name, age, department, location):
        super().__init__(name, age) # initialise the parent object
        self.department = department # set attribute only associate with
this child class
        self.location = location # set attribute only associate with this
child class

    def welcome(self): # overrides the welcome function of the parent
class
        print(f'Hi {name}, you will be teaching in the {department}')

student_a = Student('toby', 19, 'Data Engineering')
student_a.welcome()
student_a.increment_age()
student_a.welcome()

instructor_a = Instructor('Gerald', 58, 'Engineering', 'Harvard')
instructor_a.welcome()
instructor_a.increment_age()
instructor_a.welcome()
```