HOMEWORK WEEK 3

This week's homework is a research based one. You'll need to conduct independent learning, in combination with existing material (where available), to answer the questions below. The reason for this homework is to ensure you are aware of critical topics in CS. These topics were difficult to cover within the existing lesson schedules, but due to their importance are placed within the homework instead. Make sure to research, learn and then answer the following:

1. What is OOP? How may you have already made use of it (e.g. class components)?
    a. *Feel free to give a fairly light answer here - as you'll need to do the deep-part / actual meat in the following questions when you cover each of OOP's pillars*

    Object-oriented programming is a method of programming pattern based on a concept of classes and objects. An object is a self-contained component with unique attributes and behaviours. A class is a blueprint for creating an object, providing initial values for state (member variables or attributes) and implementation of behaviour.

    During the course we have briefly learned of class components in React and used them to talk about stacks, queues and linked lists.

2. What is Polymorphism?

    Polymorphism, deriving from a Greek word meaning "many-shaped", describes situations in which something occurs in several different forms. A method can process objects in a different way depending on the class type or data type.

    When working with classes that inherit the attributes and methods of a parent class, using method overriding polymorphism allows correcting inherited method to one that fits child's requirements.

    Example:

```
class Troll:
   def __init__(self):
     print("There are different types of trolls in Moominvalley")

   def house(self):
     print("Most of the trolls have a house, but not all.")

class Moomin(Troll):
   def house(self):
      print("Moomin lives in a house.")

class Snufkin(Troll):
   def house(self):
      print("Snufkin lives in a tent.")
print('Troll')
obj_troll = Troll()
obj_troll.house()

print('Moomin')
obj_moomin = Moomin()
obj_moomin.house()

print('Snufkin')
obj_snufkin = Snufkin()
obj_snufkin.house()
```

Terminal output:

```
Troll
There are different types of trolls in Moominvalley
Most of the trolls have a house, but not all.
Moomin
There are different types of trolls in Moominvalley
Moomin lives in a house.
Snufkin
There are different types of trolls in Moominvalley
Snufkin lives in a tent.
```

Polymorphism exists also in build-in methods. For example method len() will take a string and calculate the number of characters in it, but for a list it will calculate the number of items in this list.

3. What is Abstraction?

Abstraction focuses on hiding the internal implementations of a process or method from user, so that they know what they are doing, but don’t know how it is done. The irrelevant data specified in the project is hidden, and that reduces complexity and gives value to the efficiency.

In Python, abstraction is made using Abstract classes and their methods in the code.

Example:

```
from abc import ABC, abstractmethod
class Absclass(ABC):
    def print(self,x):
        print("Passed value: ", x)
    @abstractmethod
    def task(self):
        print("We are inside Absclass task")

class test_class(Absclass):
    def task(self):
        print("We are inside test_class task")
```

```
class example_class(Absclass):
   def task(self):
      print("We are inside example_class task")

#object of test_class created
test_obj = test_class()
test_obj.task()
test_obj.print(100)

#object of example_class created
example_obj = example_class()
example_obj.task()
example_obj.print(200)
```

Terminal output:

```
We are inside test_class task
Passed value:  100
We are inside example_class task
Passed value:  200
```

4. What is Inheritance?

   Inheritance is a way of creating a new class (derived/child class) for using details of an existing one (base/parent class) without modifying it. As objects are often very similar and share common logic but are not entirely the same. Inheritance helps with reusing common logic and extracting unique into separate class. The child class reuses all fields of the parent's class, but also can implement its own.

   Example:

```
# parent class
class Animal:

   def __init__(self):
      print("This is an animal")

   def whatisIt(self):
      print("Animal")

   def eat(self):
      print("Always hungry")

# child class
class Cat(Animal):

   def __init__(self):
      # call super() function
      super().__init__()
      print("This is a cat")

   def whatisIt(self):
      print("Cat")

   def sleep(self):
      print("Sleeps a lot")
```

```
cat = Cat()
cat.whatisIt()
cat.eat()
cat.sleep()
```

Terminal output:

```
This is an animal
This is a cat
Cat
Always hungry
Sleeps a lot
```

5. What is encapsulation?

   Encapsulation means binding the code and data together to create a single unit.

   Encapsulation can also refer to a mechanism where direct access to some components of an object is restricted and only a list of public functions (methods) is accessible from outside.

   Example:

```
class Person:
   def __init__(self, name, age=0):
      self.name = name
      self.__age = age

   def display(self):
      print(self.name)
      print(self.__age)

person = Person('Frank', 73)
# using class
print('Accessing from class')
person.display()
# accessing from outside
print('Accessing from outside')
print(person.name)
print(person.__age)
```

Terminal output:

```
Accessing from class
Frank
73
Accessing from outside
Frank
Traceback (most recent call last):
  File ".\person.py", line 17, in <module>
    print(person.__age)
AttributeError: 'Person' object has no attribute '__age'
```

6. What is:

    a. Agile development?

Agile development is an iterative, team-based approach. It focuses on rapid delivery of an application in complete, functional components. Instead of a general timeline for the whole project, it is delivered in 'time-boxed' phases called sprints. There are several core principles followed by agile development:
- adaptability: because of iterations, design, architecture, requirements and deliverables can change;
- customer involvement: because of the adaptability, there is a close collaboration between the customer and the development team;
- lean development: the end product should be as simple as possible;
- teamwork: team members work closely together, trying to always improve effectiveness;
- time: time-boxes sprints instead of one continuous timeline;
- sustainability: delivering software in a sustainable pace, rather than pushing for faster deadlines;
- testing: testing through every phase of the project.

Benefits:
- involvement of stakeholders;
- adaptability;
- higher quality and user-friendly products;
- flexible deliverables

Drawbacks:
- potential for a higher cost and longer deadline;
- communication must be at a high level, otherwise can lead to problems;
- intense commitment from the whole team for the project duration.

    b. Waterfall development?

Waterfall development follows a several distinct phases. They follow in sequential order, and each has to be completed before another can be started.

Phases of waterfall:
- conception: this is when the project is decided upon;

- initiation & analysis: gathering and documenting requirements for the project (including system and software)
- design: creating project's architecture and design;
- construction & coding: coding each unit (testing it along the way), and integrating units according to the design;
- testing: testing software system-wide (can include user testing, bug testing, fixes and issues)
- implementation: delivery of the finished product.

Benefits:
- clear framework;
- documentation (each phase is well documented);
- shared load (not necessarily whole team is involved);

Drawbacks:
- potentially difficult to make changes;
- last-minute testing;
- less customer involvement (this could be a bad or a good thing, depending on the project)

c. How do they differ? Which is suited for which situation?

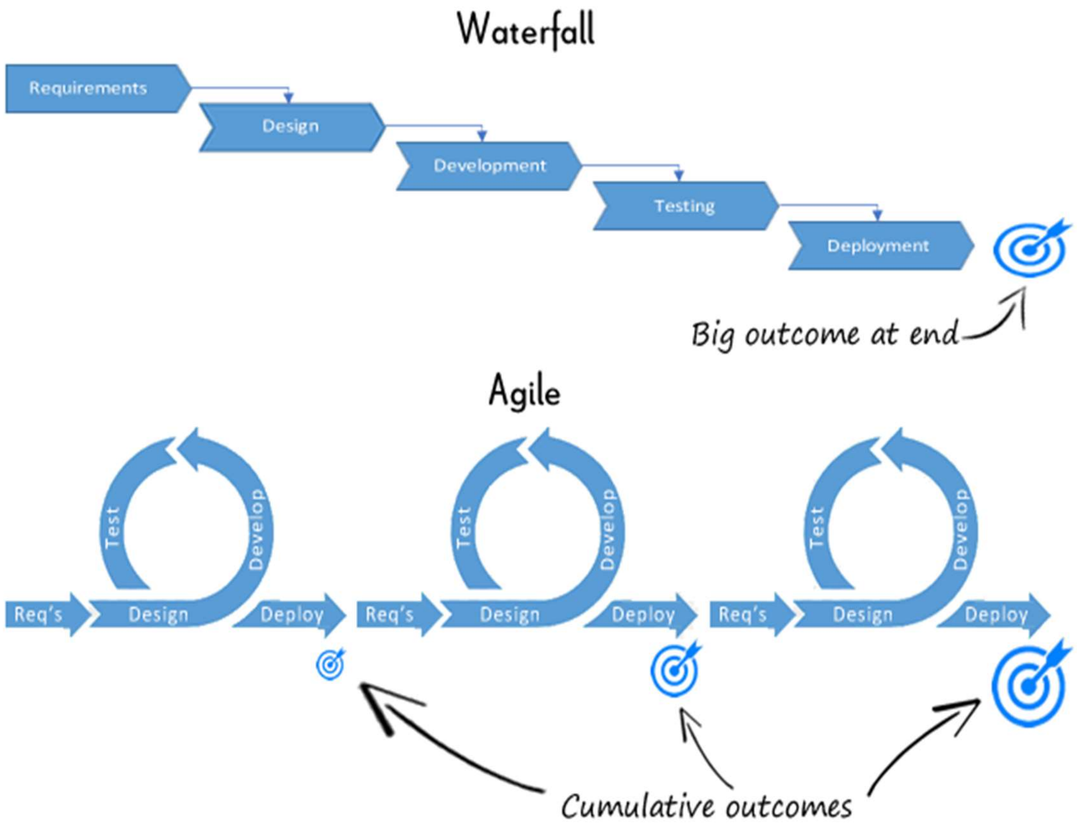

Figure 1: Waterfall vs Agile, source: https://customerthink.com/agile-versus-waterfall-for-crm-implementation-success/

Differences between methods:
- agile is iterative, waterfall sequential;

- one timeline in waterfall, time-boxed sprints in agile;
- more customer involvement in agile, less in waterfall;
- testing during creation of components in agile, testing at the end of waterfall;
- more team involvement in agile, not necessarily the same amount in waterfall

Waterfall development is best suited for organisations that have established, organized methods of development. It can be also better for delivering a project faster. It is good for software that is not rapidly changing and when there's limited or no access to the customer for providing constant feedback.

Agile development is great if a project needs to be flexible, with possibility of making changes based on customer's response to sprints.