1. Examine what it means for a program to be object-oriented.

Object-oriented programming (OOP), according to Gillis (2021), is a style of computer programming that organises software architecture around data and objects rather than functions and logic. In addition, an object is an instance of a class, which serves as a blueprint or template for the potential attributes and methods that might be present in an object to produce reusable, modular, and maintainable code (Doherty, 2020).

Utilising classes, objects, encapsulation, inheritance, and polymorphism, the four main building blocks, is a crucial part of OOP, according to Nick (2021). For each of these classes, for instance, below is an explanation:

1. **Encapsulation**: This programming approach conceals implementation information and considerably lessens software development's complexity. With this method, it is possible to update or maintain a class without worrying about the methods that depend on it. Additionally, encapsulation makes sure that your data is protected from outside tampering. Another name for encapsulation is data-hidden. Encapsulation can be considered a shield that prevents outside code from accessing data. Encapsulation enforces modularity by tying data and codes together as a single entity.

2. **Inheritance**: Regarding inheritance, classes and methods are compared to parents and children. Some characteristics of the parents may be present at birth in the child. Just as many children might inherit the attributes of their parents, inheritance ensures the reuse of codes. We search for a superclass with all or part of the code we need to implement a function, method, or class. From the existing class, we can then create our own.

3. **Polymorphism:** Polymorphism refers to having several forms. In Python, variables, functions, and objects can have many forms. Run-time polymorphism and compile-time polymorphism are the two different types of polymorphism. While the application operates, run time may take a different form, and compile time may take a different form during compilation.

4. **Abstraction**: In Python, abstraction is a programming technique in which the user is only shown the information necessary to understand the programme. Abstraction is more interested in concepts than actual happenings. In Python, abstraction is accomplished using either Abstract classes or Interfaces, showing the user the crucial details alone.

The primary principles of object-oriented programming are simplicity, code reusability, extendibility, and security. Object-oriented programming offers a powerful technique to organise and structure code, making developing and maintaining large-scale software systems easier. Encapsulation, abstraction, inheritance, and polymorphism are used to accomplish these. Reusing code ensures that programmes are produced more quickly.

2. Explore the syntax used to define a Python class.

Classes are created in Python using the 'class' keyword and the class name (GeeksforGeeks, 2019). The fundamental syntax for defining a class in Python, for instance, is as follows:

```
class ClassName:
   #class variables
   class_variable = "Hello, World!"

   #constructor
   def __init__(self, parameter1, parameter2):
       self.parameter1 = parameter1
       self.parameter2 = parameter2

   #instance method
```

```
def instance_method(self):
    print("This is an instance method.")

#static method
@staticmethod
def static_method():
    print("This is a static method.")
```

After looking at the class definition, here are some thorough explanations for each section:

1. **class ClassName:** ClassName appears after the class keyword in the first line of the class definition. The indented section of code that serves as the class's body is denoted by a colon (:).

2. **class_variable = "Hello, World!"**: A class variable is used by all class instances, such as this one. Class variables are defined outside of any methods and inside the class body.

3. **def __init__(self, parameter1, parameter2):**: When a new class instance is created, the constructor function is invoked. The phrase "init" is preceded by *two double underscores*, indicating this is a unique Python method. The *self* and *parameter1* and *parameter2* parameters are any additional arguments supplied to the constructor, and the *self* denotes the instance of the class being formed.

4. **self.parameter1 = parameter1**: This line changes the *parameter1* instance variable to the value of the *parameter1* constructor argument. Each class instance has its instance variables specified inside a method using the *self* keyword.

5. **def instance_method(self):**: This is a prime example of an instance method, often known as a method that may be used on a class instance. Since it relates to the class instance on which the method is being called, the *self* parameter is necessary for all instance methods.

6. **print("This is an instance method.")**: The *instance_method* method's body begins with this line. It merely prints a message to the console in this instance.

7. **@staticmethod**: This decorator marks the following method as static. Instead of using a class object, static methods can be called on the class itself.

8. **def static_method():**: This illustrates a static method. Because a static method is not linked to a particular class instance, it does not have a *self-parameter*.

9. **print("This is a static method.")**: The *static_method* method's body is contained in this line. Additionally, it prints a message to the console in this instance.

Despite being a simple example of a Python class, this one helps us understand the syntax and organisation of a Python class definition.

3. Investigate how to define different data types in Python.

Numerous data types that may store and modify data are included in Python (Sturtz, N.D.). For instance, the most popular Python data types and examples of how to define them are listed below:

1. Integers (**int**): Positive and negative integers are whole numbers without a decimal point. Here is a Python definition of an integer: **my_integer = 10**

2. Floating-point numbers (**float**): Decimal numbers are floating-point numbers. Here is an illustration of how to define a float in Python: **my_float = 3.14**

3. Strings (**str**): Character groups that are separated by quotation marks are called strings. Here is an illustration of how to define a string in Python:
   my_string = "Hello, World!"

4. Booleans (**bool**): Boolean values are true or false. Here is an illustration of how to define a boolean in Python: **my_boolean = True**

5. Lists (**list**): Lists are organised groups of elements from any form of data. Here is a Python definition of a list example: **my_list = [1, 2, 3, "four", 5.0]**

6. Tuples (**tuple**): Similar to lists but immutable, tuples cannot be modified after they have been defined. Here is an example of a Python tuple definition:
   my_tuple = (1, 2, 3, "four", 5.0)

7. Dictionaries (**dict**): Dictionaries are collections of key-value pairs that are not ordered. Each key has a corresponding value, which may be a value of any data type. Here is an illustration of how to define a dictionary in Python:
   my_dict = {"name": "John", "age": 30, "gender": "male"}

These are a few examples of the data types specified in Python. Sets, bytes, and complex numbers are examples of other data types. By assigning a value to a variable with the proper syntax for a specific data type, we can identify a data type in Python.

4. Define a Python class.

A class in Python serves as a guide or model for building objects with comparable attributes and behaviours (Amos, N.D.). A class specifies the properties and operations that make up an object belonging to that class. Here is a straightforward Python class definition example:

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

    def introduce(self):
        print("Hi, my name is", self.name, "and I am", self.age, "years old.")
```

In this example, we have created a class named *Person* with one method, introduce(), and the attributes *name* and *age*.
Only the __init__ method can be used when making an object from the class. The attributes of the object are initialised using this method. The *self* keyword is used in this example to set the corresponding characteristics of the object for the two arguments, *name* and *age*.
The introduced method is a standard method that writes out a message introducing the individual and accepts no arguments.

We can carry out the following actions to produce an object of the Person class:

```
person1 = Person("Hainadine", 45)
```

This generates an object of the *Person* class with the name "Hainadine" and the age of 45 called *person1*. The *introduce* method can then be called on the object:

```
person1.introduce()
```

The following message will appear as a result:

```
Hi, my name is Hainadine and I am 45 years old.
```

A Python class generally defines a template for creating objects with specific traits and behaviours. At the same time, encapsulating the data and processes unique to that entity allows for code reuse and abstraction.

References:

Gillis, A. (2021). *What is Object-Oriented Programming (OOP)?* [online] TechTarget. Available at: https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP.

Doherty, E. (2020). *What is Object Oriented Programming? OOP Explained in Depth*. [online] Educative: Interactive Courses for Software Developers. Available at: https://www.educative.io/blog/object-oriented-programming.

Nick (2021). *Polymorphism, Encapsulation, Data Abstraction and Inheritance in Object-Oriented Programming | nerd.vision*. [online] www.nerd.vision. Available at: https://www.nerd.vision/post/polymorphism-encapsulation-data-abstraction-and-inheritance-in-object-oriented-programming.

GeeksforGeeks. (2019). *Python Classes and Objects*. [online] Available at: https://www.geeksforgeeks.org/python-classes-and-objects/.

Sturtz, J. (N.D.). *Basic Data Types in Python – Real Python*. [online] realpython.com. Available at: https://realpython.com/python-data-types/.

Amos, D. (N.D.). *Object-Oriented Programming (OOP) in Python 3 – Real Python*. [online] realpython.com. Available at: https://realpython.com/python3-object-oriented-programming/.