**Questions**

1. **name all data types**

Data types are: integer, float, double, long double, bool, char

1. **Name all data structures**

Data structures are:

C++: array, set, vector, queue, priority queue, dequeue, maps, string\* (array of characters)

Python: dictionary, list, tuple, string\* (list of chars)

1. **Please describe what is object oriented programming for nontechnical person**

Object oriented programming is a kind of model or pattern that describes real-world problem using objects and interactions between them. It consist of classes that precise properties of created objects using methods and attributes.

Example 1:

Class is a blueprint of car, it define, that car has four wheels, engine, color and gearbox.

Object is instances of given blueprint with concrete engine power, blue color and automatic gearbox.

Example 2:

Class can be Animal or mammal with number of legs and given sound.

Object can be cat with four legs, and say miau, miau.

1. **What is a class**

Class:  
It's a blueprint or description for the object. It consist of attributes and methods.

1. **What is an object**

Object:  
It's an instance of the class. Object has unique name.

Eg: for class Auto, Mercedes is object of the class.

1. **What’s the difference between class and object**

**Class Object**

It's a blueprint or description for the object. The object is an instance of a class.

A class is a logical entity Object is a physical entity

A class does not allocate memory space. Object allocates memory space whenever they are created.

You can declare class only once. You can create more than one object using a class.

Example: Car. Example: Jaguar, BMW, Tesla, etc.

Class generates objects. Objects provide life to the class.

1. **Name 4 Object Oriented Programming principles**

There are 4 major principles that make an language Object Oriented.

These are Encapsulation, Data Abstraction, Polymorphism and Inheritance.

These are also called as four pillars of Object Oriented Programming.

1. **What is encapsulation**

A language mechanism for restricting direct access to some of an object’s components.

Encapsulation is achieved when each object keeps its state private, inside a class.

Other objects don’t have direct access to this state.

1. **What is inheritance**

Inheritance is a mechanism that allows one class to gain the properties of another class, in the same way, that a child inherits some attributes from each of their parents. Inheritance allows programmers to create a new class that reuses the data members and methods of an existing class**.**

1. **What is abstraction**

Its main goal is to hide unnecessary complex details of code from the user. That enables the user to implement more complex logic on top of the provided abstraction without understanding or even thinking about all the hidden complexity.

Eg: coffee machine

You don’t need to know is how the coffee machine is making a fresh cup of coffee. You don’t need to know the ideal temperature of the water, preasure or the amount of ground coffee you need to use. Someone else worried about that. You have to only press a button.

1. **What is polymorphism**

**PL:**

Polimorfizm (wielopostaciowość) umożliwia różne zachowanie tych samych metod wirtualnych w czasie wykonywania programu.

Polimorfizm to możliwość stosowania tego samego kodu dla różnych obiektów.

**ENG:**

Polymorphism is a feature of object-oriented programming languages that allows a specific algorithm to use variables of different types at different times.

It is the ability to present the same interface for several different underlying data types and different objects to respond in a unique way to the same message.

Eg:

class Ferrari:

def fuel\_type(self):

print("Petrol")

def max\_speed(self):

print("Max speed 350")

class BMW:

def fuel\_type(self):

print("Diesel")

def max\_speed(self):

print("Max speed is 240")

# normal function

def car\_details(obj):

obj.fuel\_type()

obj.max\_speed()

ferrari = Ferrari()

bmw = BMW()

car\_details(ferrari)

car\_details(bmw)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

output:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Petrol

Max speed 350

Diesel

Max speed is 240

1. **What is an interface**

PL:

Interfejs to projekt klasy z abstrakcyjnymi metodami zawierającymi jedynie nagłówki.

Interfejs służy do określenia, jakie metody muszą zostać zaimplementowane (stworzone) w klasie.

Jest to więc mechanizm podobny do klas abstrakcyjnych, z tą różnicą, że interfejs może zawierać jedynie deklaracje metod, bez ich ciał.

ENG:

Interface is as a blueprint for designing classes with only headers of abstract method. Derived class implement the interface and give concrete meaning to abstract methods from interface.

Eg:

from abc import ABC, abstractmethod

class Bank(ABC):

@abstractmethod

def balance\_check(self):

pass

@abstractmethod

def interest(self):

pass

class SBI(Bank):

def balance\_check(self):

print("Balance is 100 $")

def interest(self):

print("SBI interest is 5 $")

s = SBI()

s.balance\_check()

s.interest()

Output:

Balance is 100 $

SBI interest is 5 $

1. **What is an abstrac class**

PL:

Klasą abstrakcyjną nazywamy klasę posiadającą co najmniej jedną metodę abstrakcyjną.

Klasy abstrakcyjne nie mogą mieć swoich reprezentantów w postaci obiektów.

Klasa abstrakcyjna jest pewnym uogólnieniem innych klas (na przykład dla występujących w rzeczywistości obiektów), lecz sama jako taka nie istnieje.

Jest to więc mechanizm podobny do interfejsu, z tą różnicą, że interfejs może zawierać jedynie deklaracje metod, bez ich ciał.

ENG:

Abstract class can be a blueprint for other classes. It allows you to create methods that have to be created within any child classes built from the abstract class.

A class which contains **one** or more abstract methods is called an abstract class.

When we want to provide a common interface for different implementations of a component, we use an abstract class.

from abc import ABC, abstractmethod

class Polygon(ABC):

@abstractmethod

def noofsides(self):

pass

class Triangle(Polygon):

# overriding abstract method

def noofsides(self):

print("I have 3 sides")

class Pentagon(Polygon):

# overriding abstract method

def noofsides(self):

print("I have 5 sides")

class Quadrilateral(Polygon):

# overriding abstract method

def noofsides(self):

print("I have 4 sides")

# Driver code

R = Triangle()

R.noofsides()

K = Quadrilateral()

K.noofsides()

R = Pentagon()

R.noofsides()

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

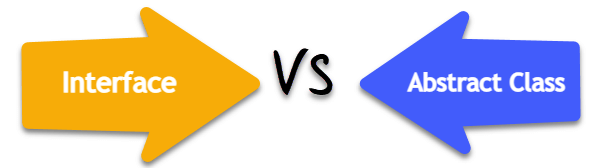
**Output:**

I have 3 sides

I have 4 sides

I have 5 sides

1. **What’s the difference between interface and abstract class**



**PL:**

Interface:

1) **Możemy dziedziczyć po wielu interfejsach.**

2) Wszystkie elementy interfejsu muszą być zaimplementowane

3) Klasa implementująca ma wszystkie elementy publiczne

4) Interfejs nie może mieć konstruktora

5) W interfejsie tworzymy jedynie deklarację metod (ich nagłówki)

Abstract class

1) **Możemy dziedziczyć tylko po jednej klasie abstrakcyjnej**.

2) Klasa dziedzicząca może implementować część metod, wszystkie metody lub żadną

3) Elementy klasy abstrakcyjnej mogą być private, protected lub public

4) Możemy zaimplementować konstruktor

5) Możemy zadeklarować i zaimplementować metody i właściwości klasy

**ENG:**

Interface:

1) In Interface, a class can implement multiple interfaces, 2) In Interface does not have access modifiers. Everything defined inside is public.

3) The Interface cannot contain data fields. It has only headers of methods.

Abstract class

1) A class can inherit only one Abstract Class.

2) Abstract Class can have an access modifier.

Private, protected or public.

3) Abstract class can have data fields and body of the methods.

When should we go for interfaces?

Since, Interfaces will not contain implemented methods, when we don’t know anything about implementation of requirements, then we should go for interfaces.

When should we go for abstract class?

An abstract class is a class which can contains few implemented methods and few unimplemented methods as well. When we know about requirements partially, but not completely, then we should go for abstract class.

When should we go for concrete class?

Concrete class is a class which is fully implemented. It contains only implemented methods. When we know complete implementation about requirements, then we should go for concrete class.

1. **Do python preserve encapsulation?**

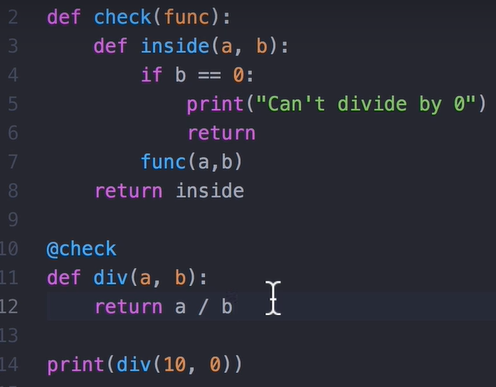
Python doesn’t have real private methods. There is only convention:

- By prefixing the name of your member with a single underscore (\_), you’re telling others “don’t touch this, unless you’re a subclass”. One underscore (\_) in the beginning of a method or attribute means you shouldn’t access this method. But this is just convention. I can still access the the variables with single underscore.

- To define Private we use double underscore (\_\_), but we can still access the private variables.

1. **What is decorator in python?**

Decorators allows programmers to modify the behaviour of a function or class. Decorators allow us to wrap another function in order to extend the behaviour of the wrapped function, without permanently modifying it.



To decorate a function having any number of argument we can use: \*args and \*\*kwargs

\*args in function definitions in python is used to pass a variable number of arguments to a function.

It is used to pass a non-key worded, variable-length argument list.

**def** funk(\*args):

print("Liczba przekazanych parametrów:", len(args))

**for** arg **in** args:

print ("Wartość:",arg)

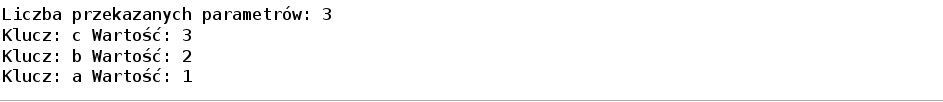
lista = [1,3,5,6]

funk(1,lista,2,'xyz',3)

\*\*kwargs in function definitions in python is used to pass a keyworded, variable-length argument list.

**def** funk(\*\*kwargs):

print("Liczba przekazanych parametrów:", len(kwargs))

**for** key, item **in** kwargs.items():

print ("Klucz:", key, "Wartość:", item)

funk(a=1,b=2,c=3)

We can use both \*args and \*\*kwargs:

**def** funk(\*args, \*\*kwargs):

print("Liczba przekazanych parametrów \*args:", len(args))

**for** arg **in** args:

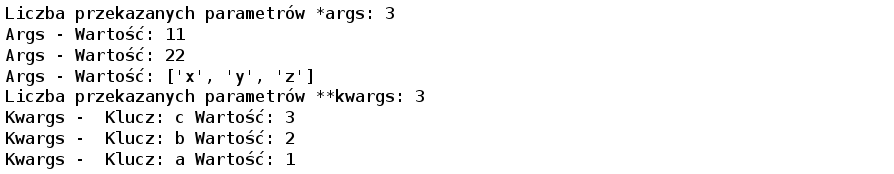
print ("Args - Wartość:", arg)

print("Liczba przekazanych parametrów \*\*kwargs:", len(kwargs))

**for** key, item **in** kwargs.items():

print ("Kwargs - ", "Klucz:", key, "Wartość:", item)

funk(11,22,['x','y','z'],a=1,b=2,c=3



1. **What is an index in databases**

An Index is a small table having only two columns. The first column have s a copy of the primary key of a table. Its second column contains a set of pointers for holding the address of the disk block where that specific key value stored.

Indexing maximize the query’s efficiency while searching. Otherwise the queries will have to search through every row to find the rows matching the conditions.

1. **Should you always use index in databases and if not - when shouldn’t you use it?**

The biggest drawback to performing the indexing database management system, you need a primary key on the table with a unique value.

Indexing Decrease performance in INSERT, DELETE, and UPDATE query.

1. **What is a race condition?**

A race condition occurs when two or more threads can access shared data and they try to change it at the same time. Because the thread scheduling algorithm can swap between threads at any time, you don't know the order in which the threads will attempt to access the shared data. Therefore, the result of the change in data is dependent on the thread scheduling algorithm, i.e. both threads are "racing" to access/change the data.

EG:

Two people tried to turn on the light using two different switches at the same time.

1. **What is a deadlock in databases?**

A deadlock happens when two processes cannot make progress because each is holding a process that the other needs. Deadlock never ever release the process.

Eg:

Process 1: has resource “A” and needs resource “B”

Proces 2: has resource “B” and needs resource “A”.

Proces 1 can’t realease resource B until gets resource A

Proces 2 can’t realease resource A until gets resource B

And both are in deadlock, each is waiting for the other.

1. **What is a block in databases?**

A block is when “A” resource is needed by two processes and one must wait on the other. But as soon as first process releases resource, second will start.

1. **What is a deadlock in multi threading?**

A deadlock is when the waiting process is still holding on to another resource that the first needs before it can finish.

Or

Deadlock occurs when two threads are waiting for each other to finish. First thread is holding another resource that the second needs and the second thread is holding resource that first thread needs.

EG:

Resource A and resource B are used by process X and process Y

X starts to use A.

X and Y try to start using B

Y 'wins' and gets B first

now Y needs to use A

A is locked by X, which is waiting for Y

1. **What is a lock in multi threating?**

A lock is when multiple processes try to access the same resource at the same time.

One process must wait for the other to finish.

EG:

When first process is running, second process have to wait some time.

But when first process finishes, second will start and everything it’s fine.

1. **What is a big O notation?**

Big O notation is tool to analyze running time of program.

We can express algorithmic complexity using the big-O notation. For a problem of N-size:

O(1) - A constant-time function/method is “order 1”, (time independent from n-elements)

O(n) - A linear-time function/method is “order n” (sum of array elements)

O(n2) - A quadratic-time function/method is “order n squared” (bubble sort)

Ο(log(n)) – Logatythmic time (search in ordered table)



Wypisane są złożoności od najwydajniejszej do najwolniejszej oraz szacowana liczba operacji, którą algorytm wykona dla 100 elementów:

O(1) - stała (od jednego do kilku operacji),

O(log n) - logarytmiczna (7 operacji),

O(n) - liniowa (100 operacji),

O(n log n) - n razy logarytm z n (700 operacji),

O(n2) - kwadratowa (10 000 operacji),

O(n3) - sześcienna (1000 000 operacji),

O(n!) - n silnia (1\*2\*3\*4\*...\*100 operacji),

O(2n) - wykładnicza (2100 operacji).