1. **Многопроцессность, многопоточность, асинхронность.**

**Конкурентность (concurrency) -** это наиболее общий термин, который говорит, что одновременно выполняется более одной задачи.

**Многопроцессность (параллельное исполнение).** Процесс – контейнер, содержащий ресурсы программы (адресное пространство, потоки, дочерние процессы).

В многопроцессорной системе процессы выполняются **параллельно** (в них присутствуют несколько системных конвейеров для исполнения команд).

В однопроцессорной системе процессы выполняются **псевдопараллельно (конкурентно)**, **т.е. процессы выполняются последовательно, занимая малые кванты процессорного времени**.

**Многопоточность** - это один из способов реализации конкурентного исполнения, при котором задача разбивается на множество потоков, занимающихся её решением.

Процесс, порождённый в операционной системе, может состоять из нескольких потоков, выполняющихся **без предписанного порядка во времени**. Контекст переключается между потоками конкурентно.

Все потоки выполняются в адресном пространстве процесса. Выполняющийся процесс имеет как минимум один (главный) поток.

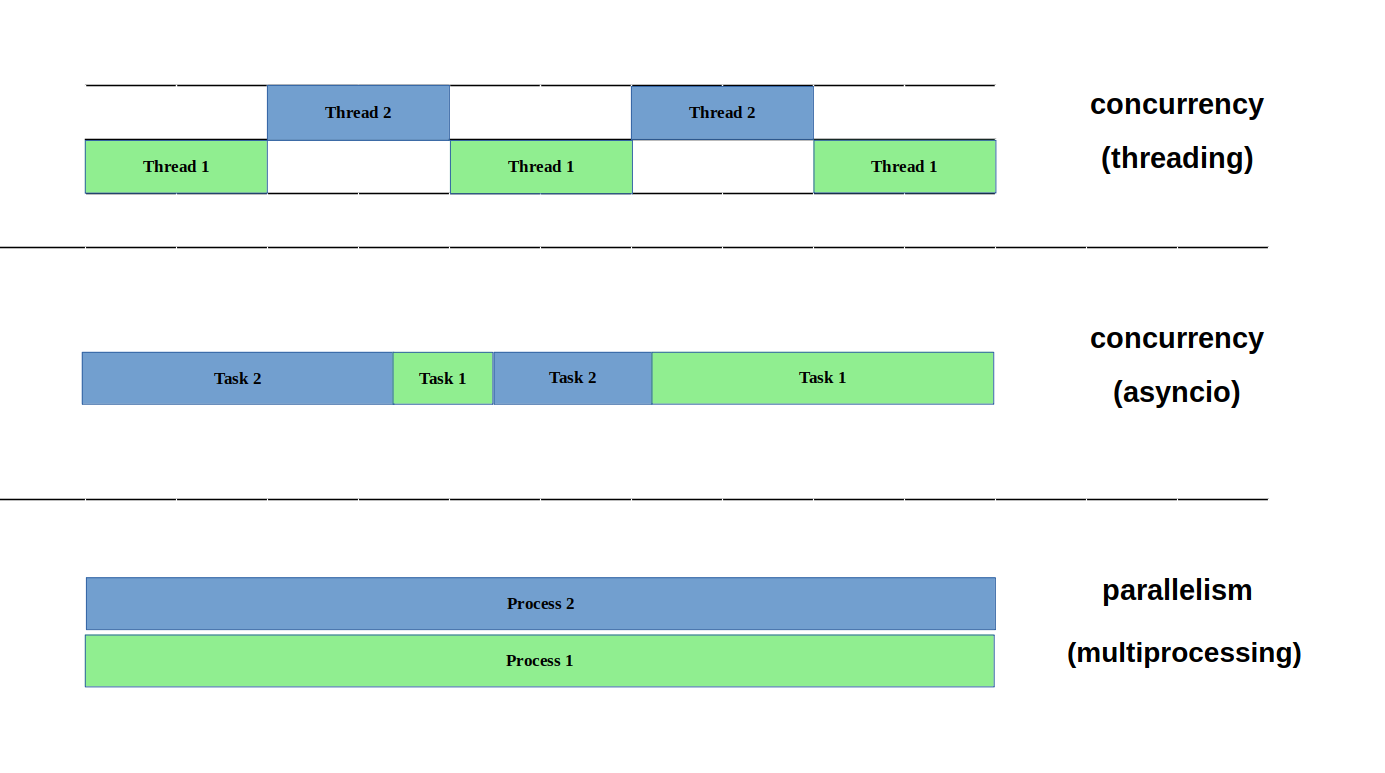
У каждого потока есть приоритет. Если есть несколько спящих потоков, которые нужно запустить, то ОС сначала запустит поток с более высоким приоритетом. Потоки с одинаковыми приоритетами запускаются в порядке очереди.

Основными проблемами при работе с многопоточностью являются **состояние гонки** и **взаимная блокировка**. Состояние гонки возникает, когда два или более потоков пытаются одновременно изменить общие данные, что может привести к непредсказуемым результатам. Взаимная блокировка — это ситуация, когда два потока блокируют друг друга, ожидая освобождения ресурсов, что приводит к "заморозке" программы. Для решения этих проблем используются различные методы синхронизации, такие как мьютексы и семафоры. В python GIL (Global Interpreter Lock).

Многопоточность прекрасно решает задачи, связанные с вводом/выводом (I/O) информации.

**Асинхронность**. Асинхронный код построен вокруг одного потока и одного процесса: он использует кооперативную многозадачность (тип многозадачности, при котором следующая задача выполняется только после того, как текущая задача явно объявит себя готовой отдать процессорное время другим задачам). Асинхронный код выполняется конкурентно.

Асинхронные подпрограммы способны вставать на паузу, ожидая получение какого-то результата и давая работать другим подпрограммам. Асинхронный код позволяет выполнять задачи одновременно. Если быть точным, то он имитирует ощущение одновременности.



1. **GIL (Global Interpreter Lock).**

Представляет собой мьютекс, который блокирует доступ к объекту Python interpreter в многопоточных средах, разрешая выполнять лишь одну инструкцию за раз.

GIL был введен не как намеренное ограничение, а скорее, как необходимая мера для обеспечения безопасности в среде многозадачности.

Один из наиболее эффективных способов обойти GIL - это использование многопроцессорной обработки вместо многозадачных потоков. Поскольку каждый процесс имеет свой собственный интерпретатор Python и собственный GIL, они могут параллельно выполняться на разных ядрах процессора.

1. **ООП**

Суть понятия объектно-ориентированного программирования в том, что все программы, написанные с применением определенных правил и критериев (парадигмы), состоят из объектов. Каждый объект — это определённая сущность со своими данными и набором доступных действий.

Класс – шаблон, на базе которого можно построить объект. Класс — это общая абстракция, которая описывает структуру объектов. Классы могут наследоваться друг от друга.

**Инкапсуляция.**

Вся информация, которая нужна для работы конкретного объекта, должна храниться внутри этого объекта. Для внешних объектов доступны только публичные атрибуты и методы.

Во многих языках частью инкапсуляции является сокрытие данных. Для этого существуют модификаторы доступа: public, private, protected.

**Наследование.**

Создание дочернего класса на основе родительского. Каждый дочерний элемент наследует методы и атрибуты родительского для дальнейшего расширения или модификации.

**Полиморфизм.**

Один и тот же метод может работать по-разному в зависимости от объекта, где он вызван, и данных, которые ему передали.

**Абстракция.**

Моделирование требуемых атрибутов и взаимодействий сущностей в виде классов для определения абстрактного представления системы.

1. **REST API**

**Протокол передачи данных** — это набор соглашений, которые определяют обмен данными между различными программами. Эти соглашения задают единообразный способ передачи сообщений и обработки ошибок.

**Транспорт** — это подмножество сетевых протоколов, с помощью которых мы можем передавать данные по сети. Такими протоколами могут быть: HTTP, AMQP, FTP.

**REST** (Representative State Transfer или Передача репрезентативного состояния) — клиент-серверная сетевая архитектура взаимодействия, при которой разработчик предоставляет упрощенный доступ клиентским приложениям пользователя к ресурсам сервера. Как правило доступ осуществляется через обмен данными в формате JSON или XML.

Для доступа клиентских приложений к данным, хранящимся на сервере, в REST API используются такие методы как POST, GET, PUT, PATCH и DELETE. При этом обмен сообщениями (вопрос-ответ) осуществляется обычно по протоколу HTTP(S).

Главная особенность REST API — обмен сообщениями без сохранения состояния (каждое сообщение самодостаточное и содержит всю информацию, необходимую для его обработки).

Чаще всего REST API использует формат JSON.

Запрос REST API от клиента к серверу всегда состоит из следующих элементов: адрес, по которому отправляется запрос, параметры (пути или запроса), заголовки (в заголовках определяется формат передаваемых данных, спецификация и версия протокола обмена и другая информация, необходимая для корректной обработки запроса) и тело запроса.

Ответ включает код ответа, заголовки и тело ответа.