1)Способность процесса выполнять несколько потоков параллельно называется многопоточностью. В идеале многопоточность может значительно улучшить производительность любой программы. И механизм многопоточности Python довольно удобен для пользователя, который вы можете быстро освоить.

**Плюсы многопоточности:**

* Multithreading может значительно повысить скорость вычислений в многопроцессорных или многоядерных системах, поскольку каждый процессор или ядро обрабатывает отдельный поток одновременно.
* Multithreading позволяет программе оставаться отзывчивой, пока один поток ожидает ввода, а другой одновременно запускает графический интерфейс. Это утверждение справедливо как для многопроцессорных, так и для однопроцессорных систем.
* Все потоки процесса имеют доступ к его глобальным переменным. Если глобальная переменная изменяется в одном потоке, она видна и другим потокам. Поток также может иметь свои собственные локальные переменные.

**Минусы многопоточности:**

* В однопроцессорной системе многопоточность не влияет на скорость вычислений. Фактически производительность системы может снизиться из-за накладных расходов на управление потоками.
* Синхронизация необходима, чтобы избежать взаимного исключения при доступе к общим ресурсам процесса. Это напрямую ведет к увеличению использования памяти и процессора.
* Многопоточность увеличивает сложность программы, что также затрудняет отладку.  
  Это повышает вероятность потенциальных ошибок.
* Это может вызвать голод, когда поток не получает регулярный доступ к общим ресурсам. Тогда он не сможет возобновить свою работу.

Python предлагает два модуля для реализации потоков в программах.

* ***thread***
* ***threading***

Если вы решили применить модуль *thread* в своей программе, используйте следующий метод для создания потоков.



Этот метод запускает новый поток и возвращает его идентификатор. Он вызовет функцию, указанную в качестве параметра «function» с переданным списком аргументов. Когда возвращается функция, поток молча завершает работу.

Здесь args - это кортеж аргументов; используйте пустой кортеж для вызова без каких-либо аргументов. Необязательный аргумент указывает словарь аргументов с ключевыми словами.

Если функция завершается с необработанным исключением, выводится трассировка стека, а затем поток выходит (это не влияет на другие потоки, они продолжают работать). Используйте приведенный ниже код, чтобы узнать больше о многопоточности.

Последний модуль ***threading*** предоставляет богатые возможности и большую поддержку потоков, чем устаревший модуль *thread*. Модуль ***threading***является отличным примером многопоточности Python.

Модуль объединяет все методы модуля *thread* и предоставляет несколько дополнительных методов.

* **threading.activeCount()**: находит общее число активных объектов потока.
* **threading.currentThread()**: его можно использовать для определения количества объектов потока в элементе управления потоком вызывающей стороны.
* **threading.enumerate()**: он предоставит вам полный список объектов потока, которые в данный момент активны.

Помимо описанных выше методов, модуль также представляет класс ***Thread***

Вы можете выполнить следующие шаги для создания нового потока с помощью модуля .

* Создайте класс наследовав его от **Thread**.
* Переопределите метод **\_\_ init \_\_ (self [, args])** для предоставления аргументов в соответствии с требованиями.
* Затем переопределите метод **run(self [, args])**, чтобы создать бизнес-логику потока.

Как только вы определили новый подкласс *Thread*, вы должны создать его экземпляр, чтобы начать новый поток. Затем вызовите метод для его запуска. В конечном итоге он вызовет метод для выполнения бизнес-логики.

В ​**синхронных** операциях задачи выполняются друг за другом. В **асинхронных** задачи могут запускаться и завершаться независимо друг от друга. Одна асинхронная задача может запускаться и продолжать выполняться, пока выполнение переходит к новой задаче. Асинхронные задачи **​не** блокируют (не заставляют ждать завершения выполнения задачи) операции и обычно выполняются в фоновом режиме

2) Tkinter – это пакет для Python, предназначенный для работы с библиотекой Tk. Библиотека Tk содержит компоненты графического интерфейса пользователя (graphical user interface – GUI), написанные на языке программирования Tcl.

Под графическим интерфейсом пользователя (GUI) подразумеваются все те окна, кнопки, текстовые поля для ввода, скроллеры, списки, радиокнопки, флажки и др., которые вы видите на экране, открывая то или иное приложение. Через них вы взаимодействуете с программой и управляете ею. Все эти элементы интерфейса вместе будем называть виджетами (widgets).

Tkinter импортируется стандартно для модуля Python любым из способов: import tkinter, from tkinter import \*, import tkinter as tk

Строка from tkinter import \* подключает модуль;

Строка root=Tk() создает окно приложения;

Строка button1=Button(root,text='Вывод текста')создает объект класса Button, то есть обычную кнопку;

Строка button1.pack() размещает эту кнопку в окне приложения. Метод pack представляет самый простой способ размещения, в дальнейшем мы узнаем и другие способы.

Строка root.mainloop() запускает программу в цикле.