1 – Kernel is central component of an operating system that manages operations of computer and hardware. It basically manages operations of memory and CPU time. It is core component of an operating system. Kernel acts as a bridge between applications and data processing performed at hardware level using inter-process communication and system calls.

The kernel consists of several essential components

1- **process management component oversees the creation, execution, and termination of processes, ensuring efficient multitasking.** It allocates resources, such as CPU time and memory, to different processes, enabling them to run smoothly.

2- **the memory management component handles the allocation and deallocation of memory to various processes.** It ensures that each process has the necessary memory to execute its instructions, preventing conflicts and optimizing memory usage.

3- **the file system component manages the organization and access to files and directories on storage devices.** It facilitates reading, writing, and modifying files, ensuring data integrity and security.

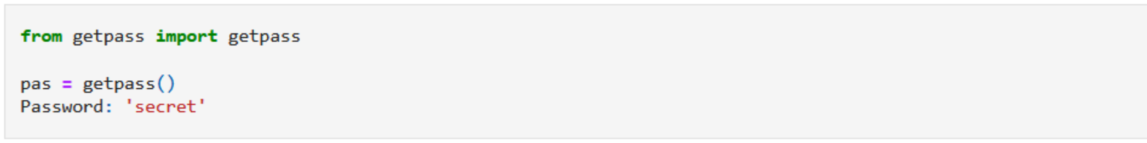
4- **the device driver component allows the kernel to communicate with hardware devices.** It provides an interface for the operating system to interact with different peripherals, such as printers, keyboards, and network adapters.

5- **the networking component enables network communication by implementing various protocols and managing network connections.**

- A Kernel is provided with a protected Kernel Space which is a separate area of memory and this area is not accessible by other application programs. So, the code of the Kernel is loaded into this protected Kernel Space. Apart from this, the memory used by other applications is called the User Space. As these are two spaces in the memory, communication between them is a bit slower.

- The Kernel is responsible for low-level tasks such as disk management, memory management, task management, etc. It provides an interface between the user and the system's hardware components. When a process makes a request to the Kernel, then it is called System Call.

2- How to hide the password while typing it in python ?



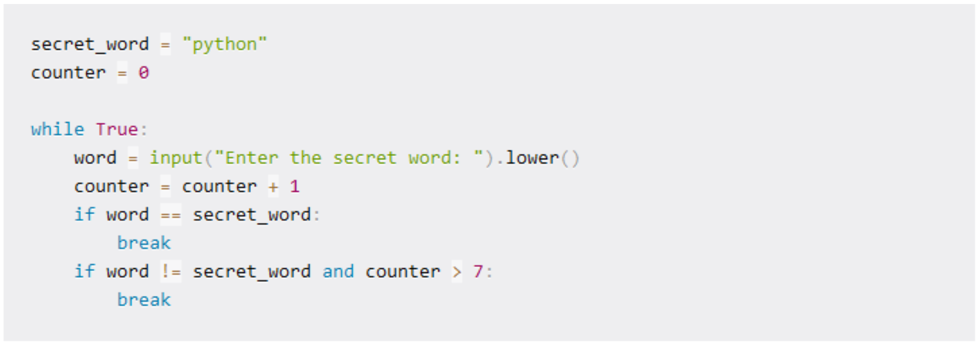
```
from getpass import getpass

pas = getpass()
Password: 'secret'
```

3- https://www.youtube.com/watch?v=VdkfdBm_6W4&ab_channel=CalebCurry

---

4- DO-While in python



```
secret_word = "python"
counter = 0

while True:
    word = input("Enter the secret word: ").lower()
    counter = counter + 1
    if word == secret_word:
        break
    if word != secret_word and counter > 7:
        break
```

The code will run at least one time, asking for user input.

It is always guaranteed to run at least once, with True, which otherwise creates an infinite loop.

If the user inputs the correct secret word, the loop is terminated.

If the user enters the wrong secret word more than 7 times, then the loop will be completely exited.

5-

```
for _ in iter(int, 1):
    pass
```

→ iter()function to call a function repeatedly until its return value matches that argument. This would loop forever as 1 will never be equal to 0.

6- In software engineering, **dependency injection** is a technique whereby one object (or static method) supplies the dependencies of another object. A dependency is an object that can be used (a service).

So, transferring the task of creating the object to someone else and directly using the dependency is called dependency injection.

There are basically three types of dependency injection:

1. **constructor injection:** the dependencies are provided through a class constructor.
2. **setter injection:** the client exposes a setter method that the injector uses to inject the dependency.
3. **interface injection:** the dependency provides an injector method that will inject the dependency into any client passed to it. Clients must implement an interface that exposes a setter method that accepts the dependency.

Libraries and Frameworks that implement DI

- Spring (Java)
- Google Guice (Java)
- Dagger (Java and Android)
- Castle Windsor (.NET)
- Unity(.NET)

---

is recursive code fast as iterator code!?

1. In a platform that does not support tail call optimization, such as Java, the recursive approach will be slower and eat up memory faster.
2. If the code is not written in such a manner that the recursive call can be made immediately prior to the return statement, then the recursive approach will be slower and more memory intensive than an iterative approach.
3. If the app is running on a multi-core system, then a purely functional language (that has compiler support for parallelization) *can* create code for a recursive approach that will be faster than an iterative approach possibly in another language. This assumes, of course, that the programmer has not taken the time to manually parallelize that iterative approach.

---

4. In Java, C, and Python, recursion is fairly expensive compared to iteration (in general) because it requires the allocation of a new stack frame. In some C compilers, one can use a compiler flag to eliminate this overhead, which transforms certain types of recursion (certain types of tail calls) into jumps instead of function calls.
5. In functional programming language implementations, sometimes, iteration can be very expensive and recursion can be very cheap. In many, recursion is transformed into a simple jump, but changing the loop variable (which is mutable) *sometimes* requires some relatively heavy operations, especially on implementations that support multiple threads of execution. The mutation is expensive in some of these environments because of the interaction between the mutator and the garbage collector, if both might be running at the same time.
6. In short, the answer depends on the code and the implementation. Use whatever style you prefer. If you're using a functional language, recursion *might* be faster. If you're using imperative language, iteration is *probably* faster.

➢ What are Clean Code Rules?

- Clean code means benefits: pleasant work, less stress, unit tests, easy debugging, and better quality.
- Follow standard convention
- Simpler is better
- Reduce complexity as much as possible
- Keep configurable data at high levels
- Prefer polymorphism to if/else or switch/case
- Separate multi-threading code
- Prevent over-configurability
- Use dependency injection

- Follow the Law of Demeter. A class should know only its direct dependencies.

  REF : https://gist.github.com/wojteklu/73c6914cc446146b8b533c0988cf8d29

Thread is a separate execution path. It is a lightweight process that the operating system can schedule and run concurrently with other threads. The operating system creates and manages threads, and they share the same memory and resources as the program that created them. This enables multiple threads to collaborate and work efficiently within a single program.

A thread is a single sequence stream within a process. Threads are also called lightweight processes as they possess some of the properties of processes. Each thread belongs to exactly one process. In an operating system that supports multithreading, the process can consist of many threads.

Types of threads : https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/4_Threads.html

Advantages of Threading

- Threads improve the overall performance of a program.
- Threads increases the responsiveness of the program
- Context switching time in threads is faster.
- Threads share the same memory and resources within a process.
- Communication is faster in threads.
- Threads provide concurrency within a process.
- Enhanced throughput of the system.