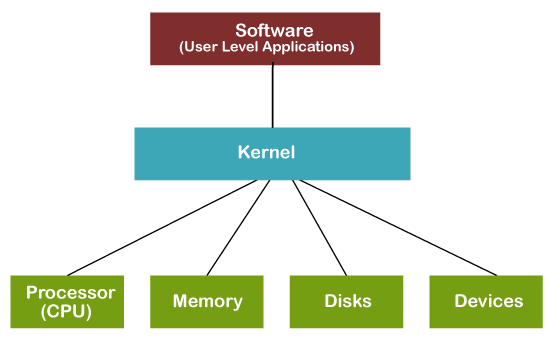


*The Second Session Tasks*

* **What is component of the kernel and how does it work and manage the operating systems**?

* 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.



* **What is Library in Python to make password Not appear?**
* **from getpass import getpass.**
* **How to make Jupyter support C++ Language?**
* It is possible in Linux and ios by xeus-cling or cling but not in windows
* A founded solution to make it possible by using binder but it needed internet.
* <https://notebooks.gesis.org/binder/jupyter/user/haneenibrahim2-binder-cpp-vv04nbeo/lab/tree/Untitled.ipynb> This first experience for me by using a binder.
* **How to make do while concept in Python?**

**count=0**

**count2=1**

**while True:**

**if count==count2:**

**print("Yes")**

**count=count+1**

**else:**

**print("Hello")**

**break**

* **a python code that passes by 2 one time and passes by times 2 another time?**

a="abcdefghigklmn

length=len(a)

for i in range(0,length):

i=i\*2

print(a[i])

* **how to make an infinite for loop in python without using extra memory?**

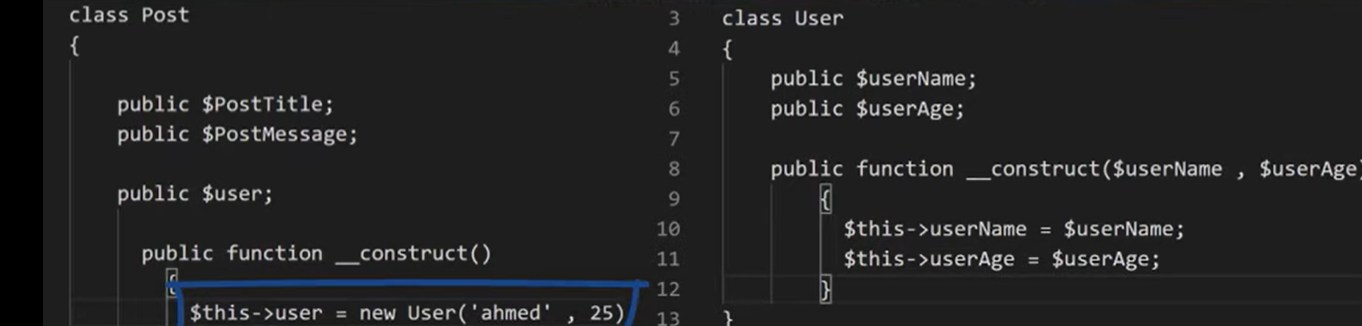
from itertools import count

for i in count(0):

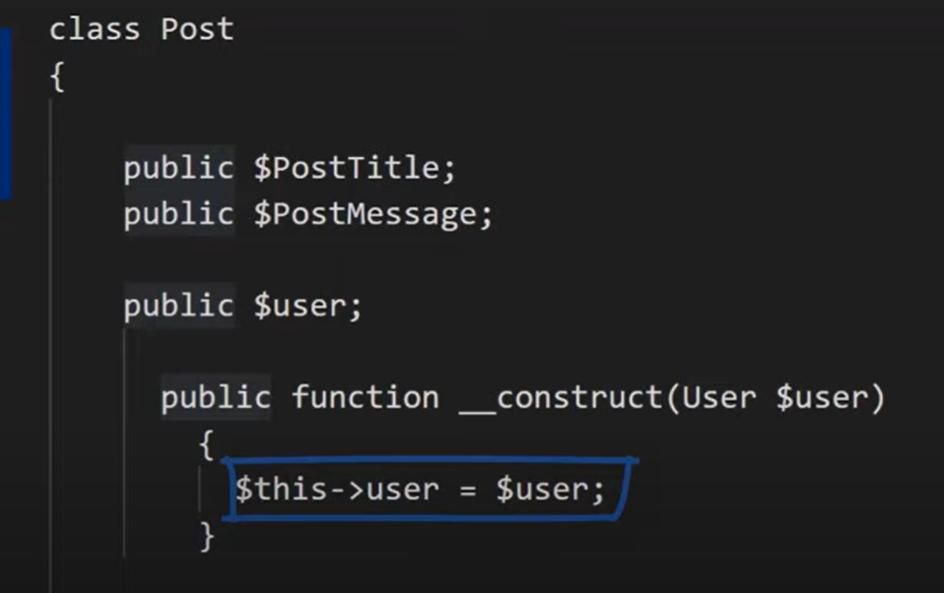
print(i)

* **What is dependency injection?**

The class post depends on the class user if we change the constructor we should edit the initialization var in class post. (**dependency )**

****

In this class, we pass an object from the class user without initialization it, this means injection.

****

* **Why is recursive code fast as iterator code?**
* 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.
* 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.
* 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.
* **What is the Threads code?**
* A thread is an execution unit that has its own program counter, a stack, and a set of registers that reside in a process. Threads can’t exist outside any process. Also, each thread belongs to exactly one process. The information like code segment, files, and data segment can be shared by the different threads.
* Threads are popularly used to improve the application through parallelism. Actually, only one thread is executed at a time by the CPU, but the CPU switches rapidly between the threads to give an illusion that the threads are running parallel.
* Threads are also known as lightweight processes.

#### Types of Thread

User-Level Thread

1. The user-level threads are managed by users and the kernel is not aware of it.
2. These threads are faster to create and manage.
3. The kernel manages them as if it was a single-threaded process.
4. It is implemented using user-level libraries and not by system calls. So, no call to the operating system is made when a thread switches the context.
5. Each process has its own private thread table to keep the track of the threads.

Kernel-Level Thread

1. The kernel knows about the thread and is supported by the OS.
2. The threads are created and implemented using system calls.
3. The thread table is not present here for each process. The kernel has a thread table to keep the track of all the threads present in the system.
4. Kernel-level threads are slower to create and manage as compared to user-level threads.

#### Advantages of threads

1. Performance: Threads improve the overall performance(throughput, computational speed, responsiveness) of a program.
2. Resource sharing: As the threads can share the memory and resources of any process it allows any application to perform multiple activities inside the same address space.
3. Utilization of Multiple Processor Architecture: The different threads can run parallel on the multiple processors hence, this enables the utilization of the processor to a large extent and efficiency.
4. Reduced Context Switching Time: The threads minimize the context switching time as in Thread Switching, and the virtual memory space remains the same.
5. Concurrency: Thread provides concurrency within a process.
6. Parallelism: Parallel programming techniques are easier to implement.

#### Difference between process and thread

1. Definition: process means a program that is currently under execution, whereas thread is an entity that resides within a process that can be scheduled for execution.
2. Termination Time: The processes take more time to terminate, whereas threads take less time to terminate.
3. Creation Time: The process creation time takes more time as compared to thread creation time.
4. Context Switching Time: Process context switching takes more time as compared to thread context switching.
5. Communication: Communication between threads requires less time as compared to communication between processes.
6. Resources: Processes are also called heavyweight processes as they use more resources. The threads are called lightweight processes as they share resources.
7. Memory: A Process is run in separate memory space, whereas threads run in shared memory space.
8. Sharing Data: Different processes have different copies of data, files, and codes whereas threads share the same copy of data, file, and code segments.