SCHEDULING

Schedule LIbrary

Schedule module lets developer run Python functions (or any other callable) periodically at predetermined intervals using a simple syntax.

Schedule Library is used to schedule a task at a particular time every day or a particular day of a week. We can also set time in 24 hours format that when a task should run. Basically, Schedule Library matches the systems time.

Installation:

```
pip install schedule
```

```
Use: import schedule
```

```python
import schedule

cnt=0
def main():
    global cnt
    cnt+=1
    print('Python script is running with count',cnt)

schedule.every(10).seconds.do(main)
while True:
    schedule.run_pending()
```

schedule.every(10).seconds.do(main) : Scheduled at every 10 secs, and call main method.
at(time_str) : Schedule at specific time

run_pending() : Runs on the default scheduler time. Run all jobs that are scheduled to run.
run_all(delay_seconds=0) : Runs all jobs regardless if they are scheduled or not.

```
# After every 15 mins main() is called.
schedule.every(15).minutes.do(main)

# After every hour main() is called.
schedule.every().hour.do(main)

# After every week main() is called.
schedule.every().week.do(main)

# Every day at 12am or 00:00 time bedtime() is called.
schedule.every().day.at("00:00").do(bedtime)

# Every monday good_luck() is called
schedule.every().monday.do(good_luck)

# Every tuesday at 18:00 main() is called
schedule.every().tuesday.at("18:00").do(main)
```

CACHING IN PYTHON

Cache

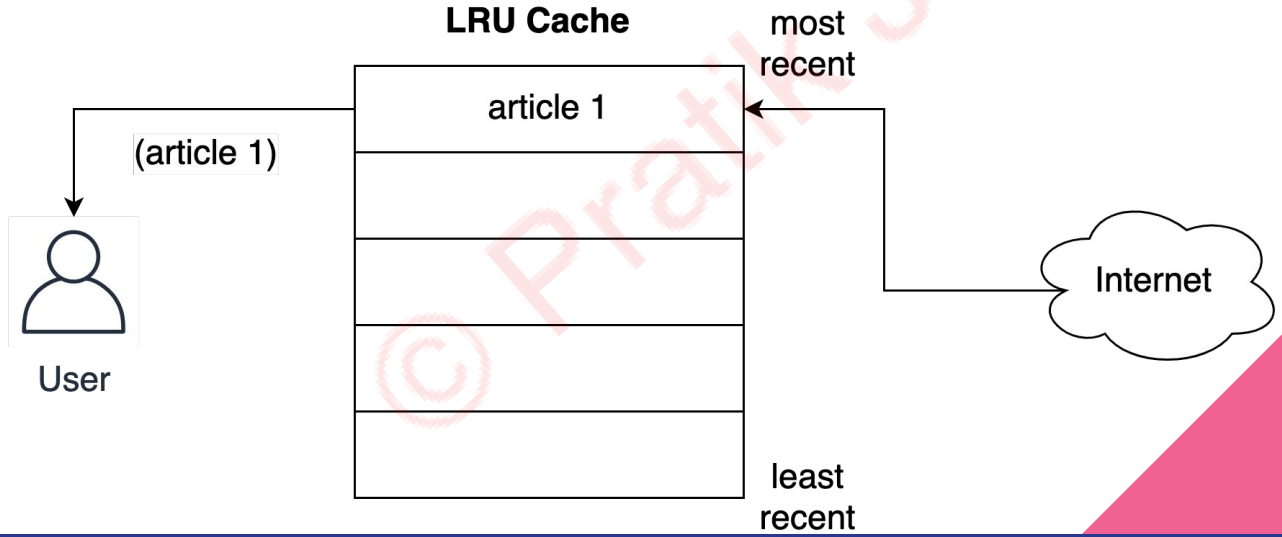
A cache is a special storage space for temporary files that makes a device, browser, or app run faster and more efficiently.

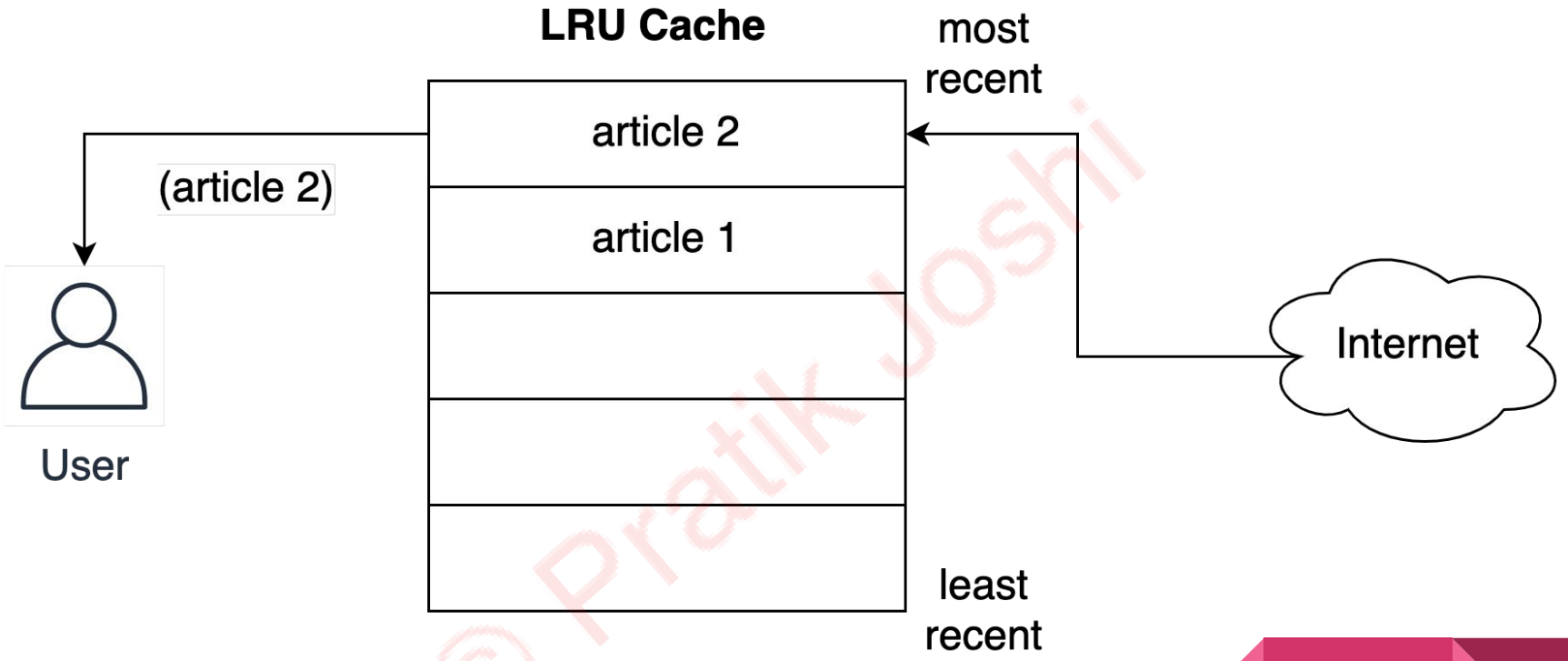
A cache is a reserved storage location that collects temporary data to help websites, browsers, and apps load faster. Whether it's a computer, laptop or phone, web browser or app, there are some variety of a cache.

A cache makes it easy to quickly retrieve data, which in turn helps devices run faster. It acts like a memory bank, making it easy to access data locally instead of redownloading it every time you visit a website or open an app.

Least Recently Used (LRU) Cache

A cache implemented using the LRU strategy organizes its items in order of use. Every time you access an entry, the LRU algorithm will move it to the top of the cache. This way, the algorithm can quickly identify the entry that's gone unused the longest by looking at the bottom of the list.





lru_cache()

lru_cache() is a function in **functools** module which helps in reducing the execution time of the function by using memoization technique.

Use:

```
from functools import lru_cache
```

Syntax:

```
@lru_cache(maxsize=128, typed=False)    (Used as Decorator)
```

Parameters:

Maxsize:

This parameter sets the size of the cache, the cache can store upto maxsize most recent function calls, if maxsize is set to None, the LRU feature will be disabled and the cache can grow without any limitations

typed:

If typed is set to True, function arguments of different types will be cached separately. For example, f(3) and f(3.0) will be treated as distinct calls with distinct results and they will be stored in two separate entries in the cache

Use Case Of LRU Cache

```
#Function Without Cache
from functools import lru_cache
import time

def fib_without_cache(n):
    if n < 2:
        return n
    return fib_without_cache(n - 1) + fib_without_cache(n - 2)

begin = time.time()
res=fib_without_cache(35)
print(res)
end = time.time()
print("Time taken to execute the function without lru_cache is", end - begin)


...
...
```

```
#Function With Cache
@lru_cache(maxsize=128)
def fib_with_cache(n):
    if n < 2:
        return n
    return fib_with_cache(n - 1) + fib_with_cache(n - 2)

begin = time.time()
res=fib_with_cache(35)
print(res)
end = time.time()
print("Time taken to execute the function with lru_cache is", end - begin)
```

Op:

Time taken to execute the function without lru_cache is 2.0684657096862793

Time taken to execute the function with lru_cache is 0.002

There is a significant difference in time between function with cache and without cache.