

```
File Home Insert Draw Design Layout References Mailings Review View Help Acrobat Comments Editing Share
Aptos (Body) 12
Windows PowerShell
PS E:\> cd .\COS20007\
PS E:\COS20007> cd .\2025-HX05-COS20007-Object-Oriented-Programming\
PS E:\COS20007\2025-HX05-COS20007-Object-Oriented-Programming> cd .\Clock_python\
PS E:\COS20007\2025-HX05-COS20007-Object-Oriented-Programming\Clock_python> python .\main.py
Before reset:
Name is Counter 1
Tick is 9
Name is Counter 2
Tick is 14
Name is Counter 1
Tick is 9

After reset:
Name is Counter 1
Tick is 0
Name is Counter 2
Tick is 14
Name is Counter 1
Tick is 0

Clock demo:
11:59:59
12:00:00
12:00:01
12:00:02
12:00:03

Total memory usage: 1.77 KB
Peak memory usage: 2.20 KB
PS E:\COS20007\2025-HX05-COS20007-Object-Oriented-Programming\Clock_python>
```

clock.py main.py counter.py

```
TestCounter TestCounter
Windows PowerShell
PS E:\COS20007\2025-HX05-COS20007-Object-Oriented-Programming\Clock\SimpleApp> cd "E:\COS20007\2025-HX05-COS20007-Object-Oriented-Programming\Clock\MemoryTest"
PS E:\COS20007\2025-HX05-COS20007-Object-Oriented-Programming\Clock\MemoryTest> dotnet run
=== Simple Clock Application with Memory Monitoring ===

Before Reset:
Name is Counter 1
Tick is 9
Name is Counter 2
Tick is 14
Name is Counter 1
Tick is 9

After Reset:
Name is Counter 1
Tick is 0
Name is Counter 2
Tick is 14
Name is Counter 1
Tick is 0

=== Memory Usage Statistics ===
Managed heap memory: 65,944 bytes (0.06 MB)
Process working set: 22,196,224 bytes (21.17 MB)
Memory after GC: 63,200 bytes (0.06 MB)
Available system memory: 16,856,203,264 bytes (16075.33 MB)

Press any key to exit...
```

from counter import Counter

class Clock:

```

def __init__(self):
    self.id = "SWS01358"

    self.is_12hr = self._is_12_hour_format()

    self.hour = Counter("Hour")
    self.min = Counter("Minute")
    self.sec = Counter("Second")

def _is_12_hour_format(self):
    last_char = self.id[-1]

    return last_char.isdigit() and int(last_char) <= 5

def tick(self):
    self.sec.increment()

    if self.sec.ticks >= 60:
        self.sec.reset()

        self.min.increment()

        if self.min.ticks >= 60:
            self.min.reset()

            self.hour.increment()

            if self.hour.ticks >= (12 if self.is_12hr else 24):
                self.hour.reset()

def set_time(self, h, m, s):
    for _ in range(h):
        self.hour.increment()

    for _ in range(m):

```

```
        self.min.increment()

    for _ in range(s):

        self.sec.increment()


    def reset(self):

        self.hour.reset()

        self.min.reset()

        self.sec.reset()


    def __str__(self):

        return f"{self.hour.ticks:02}:{self.min.ticks:02}:{self.sec.ticks:02}"
```

```
class Counter:

    def __init__(self, name):

        self._count = 0

        self._name = name


    def increment(self):

        self._count += 1


    def reset(self):

        self._count = 0
```

```
@property
```

```
def name(self):
```

```
    return self._name
```

```
@name.setter
```

```
def name(self, value):
```

```
    self._name = value
```

```
@property
```

```
def ticks(self):
```

```
    return self._count
```

```
@property
```

```
def reset_by_default(self):
```

```
    return 2147483647358 # just for compatibility, not actually used
```

```
import tracemalloc
```

```
from counter import Counter
```

```
from clock import Clock
```

```
def print_counters(counters):
```

```
    for counter in counters:
```

```
        print(f"Name is {counter.name}")
```

```
        print(f"Tick is {counter.ticks}")
```

```
def main():

    # Starting the monitoring
    tracemalloc.start()

    my_counters = [None] * 3
    my_counters[0] = Counter("Counter 1")
    my_counters[1] = Counter("Counter 2")
    my_counters[2] = my_counters[0] # same reference as counter 1

    for _ in range(9):
        my_counters[0].increment()
    for _ in range(14):
        my_counters[1].increment()

    print("Before reset:")
    print_counters(my_counters)

    my_counters[2].reset()

    print("\nAfter reset:")
    print_counters(my_counters)

    # Optional Clock demo
    print("\nClock demo:")
    clock = Clock()
    clock.set_time(11, 59, 58)
```

```
for _ in range(5):
```

```
    clock.tick()
```

```
    print(clock)
```

```
# Display total memory usage
```

```
current, peak = tracemalloc.get_traced_memory()
```

```
print(f"\nTotal memory usage: {current / 1024:.2f} KB")
```

```
print(f"Peak memory usage: {peak / 1024:.2f} KB")
```

```
# Stopping the library
```

```
tracemalloc.stop()
```

```
if __name__ == "__main__":
```

```
    main()
```