香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

# Introduction to Computer Science: Programming Methodology

# Lecture 6 Object-Oriented Programming (Additional Note)

**Prof. Yunming XIAO**

**School of Data Science**

# Mutable objects

```
from Circle import Circle

def main():
    # Create a Circle object with radius 1
    myCircle = Circle()

    # Print areas for radius 1, 2, 3, 4, and 5
    n = 5
    printAreas(myCircle, n)

    # Display myCircle.radius and times
    print("\nRadius is", myCircle.radius)
    print("n is", n)

# Print a table of areas for radius
def printAreas(c, times):
    print("Radius \t\tArea")
    while times >= 1:
        print(c.radius, "\t\t", c.getArea())
        c.radius = c.radius + 1
        times = times - 1

main() # Call the main function
```

```
Radius                  Area

1                       3.141592653589793
2                       12.566370614359172
3                       29.274333882308138
4                       50.26548245743669
5                       79.53981633974483

Radius is 6
n is 5
```

# Practice

```
class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    n = 1
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c = Count(5)
    n = 3

main() # Call the main function
```

- What would be the output of the above program?

# Diving into mutable/immutable objects (I)

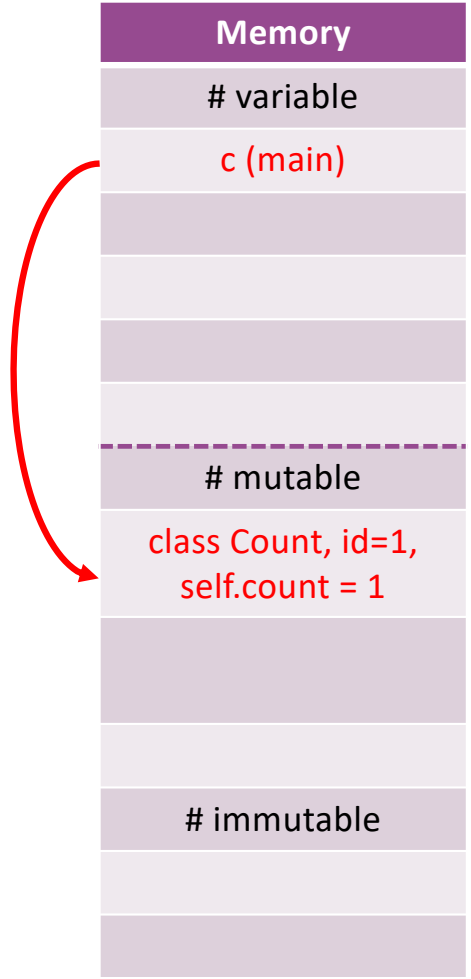
```
class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    n = 1
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c = Count(5)
    n = 3

main() # Call the main function
```



# Diving into mutable/immutable objects (I)

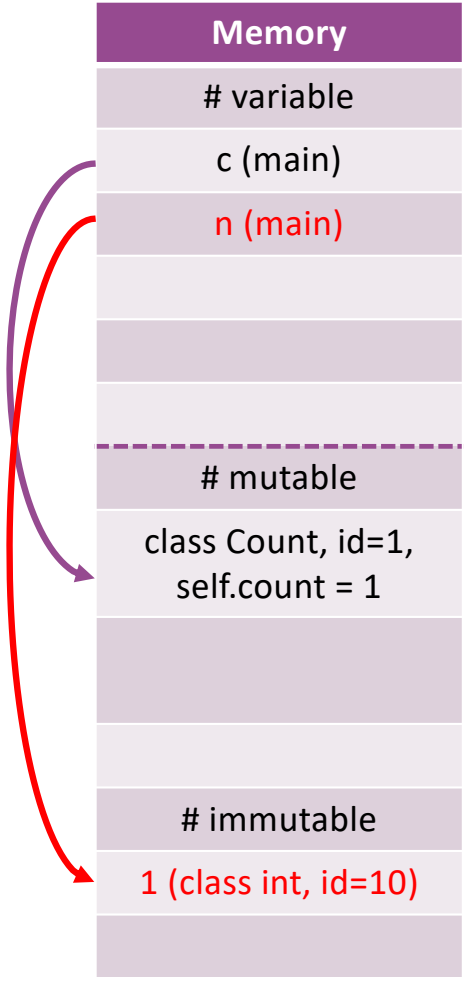
```
class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    n = 1
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c = Count(5)
    n = 3

main() # Call the main function
```



# Diving into mutable/immutable objects (I)

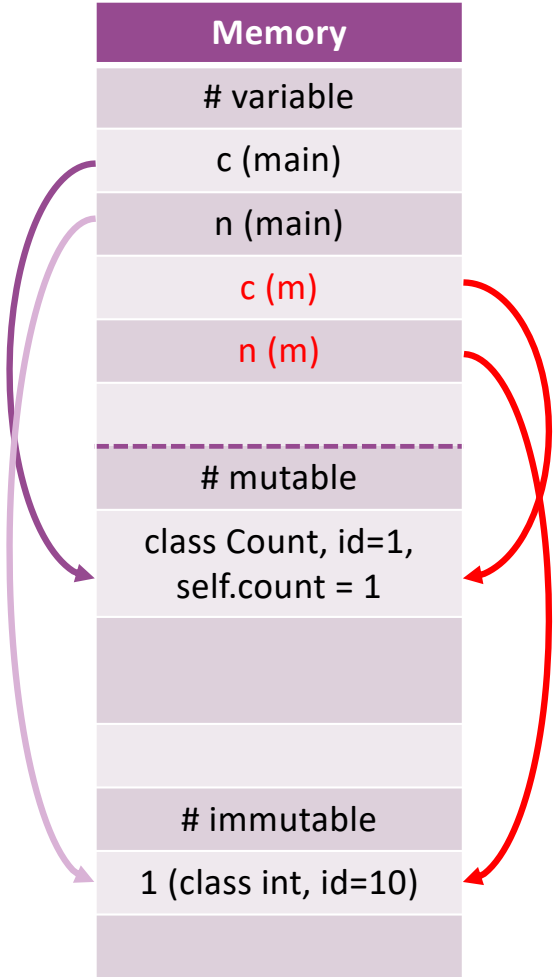
```
class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    n = 1
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c = Count(5)
    n = 3

main() # Call the main function
```

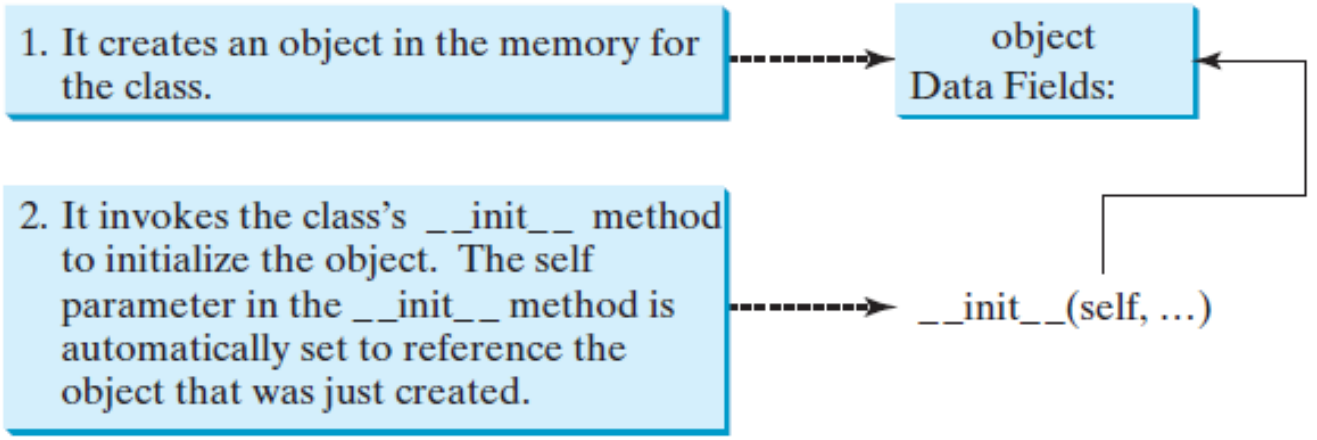


# Revisit: constructing objects

- Once a class is defined, you can create objects from the class with a constructor. The constructor does two things:

✓It creates an object in the memory for the class

✓It invokes the class's __init__() method to initialize the object



# Diving into mutable/immutable objects (I)

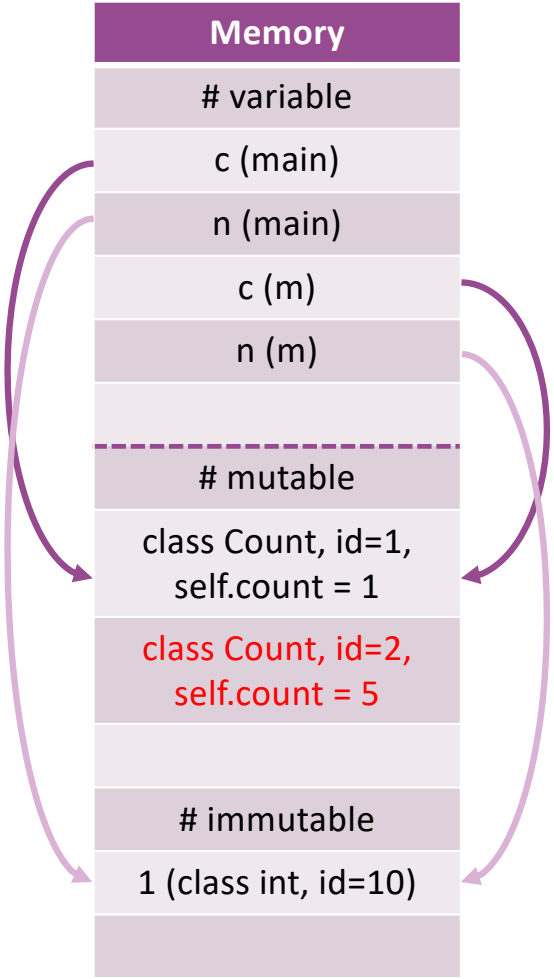
```
class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    n = 1
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c = Count(5)
    n = 3

main() # Call the main function
```



# Diving into mutable/immutable objects (I)

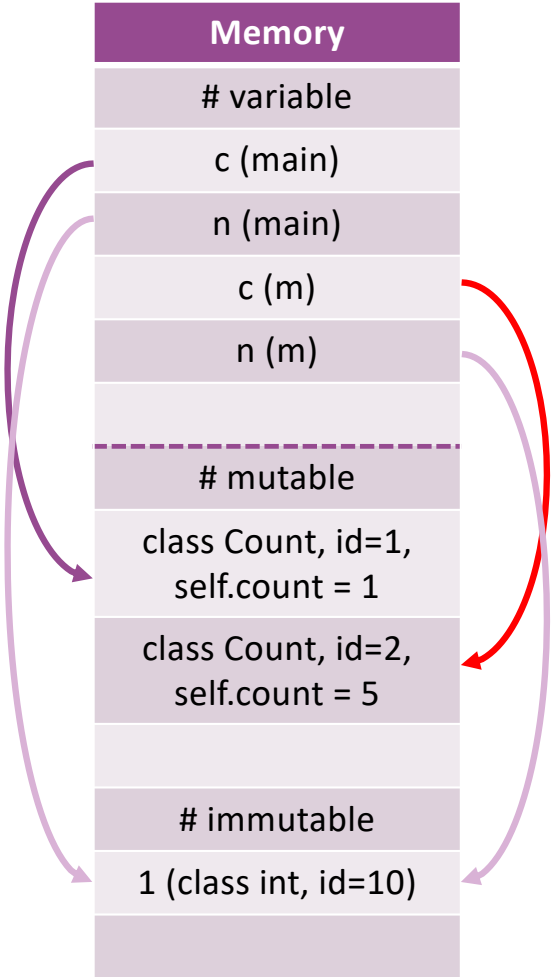
```
class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    n = 1
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c = Count(5)
    n = 3

main() # Call the main function
```



# Diving into mutable/immutable objects (I)

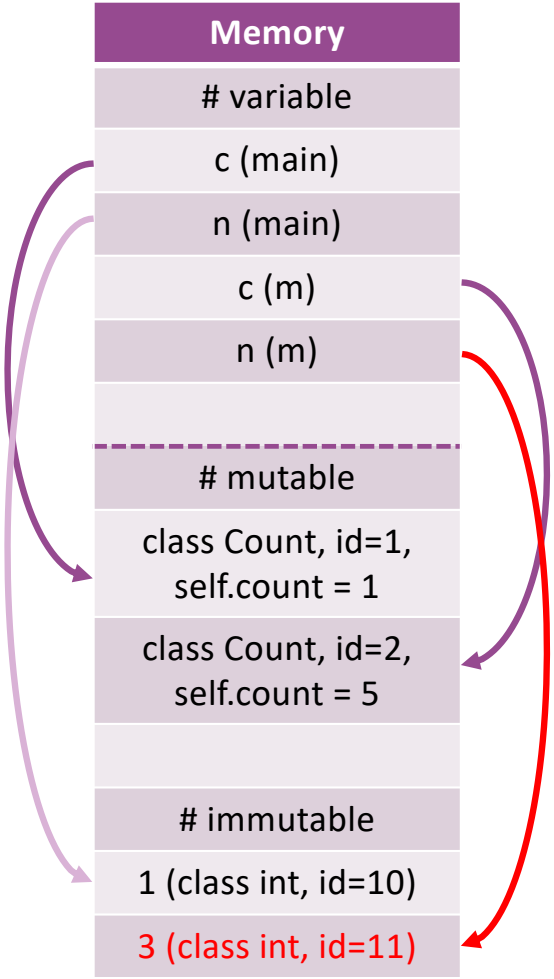
```
class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    n = 1
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c = Count(5)
    n = 3

main() # Call the main function
```



# Diving into mutable/immutable objects (I)

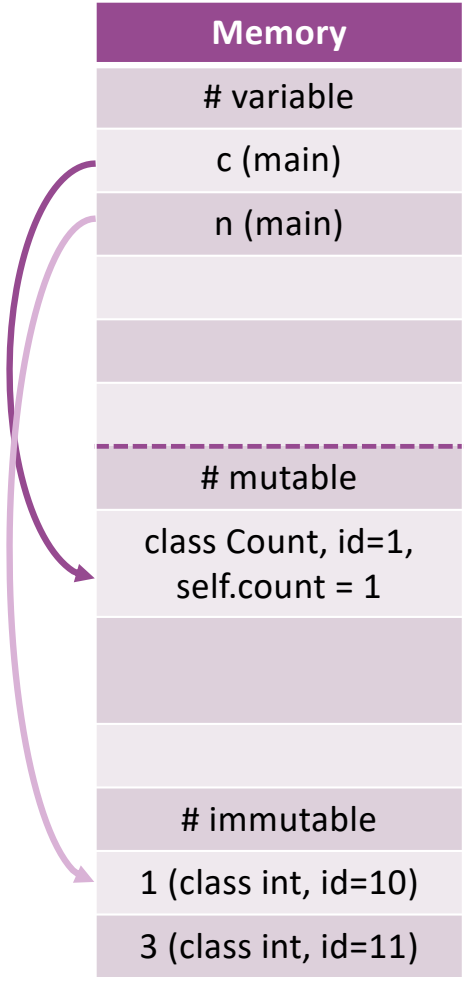
```
class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    n = 1
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c = Count(5)
    n = 3

main() # Call the main function
```



# Diving into mutable/immutable objects (I)

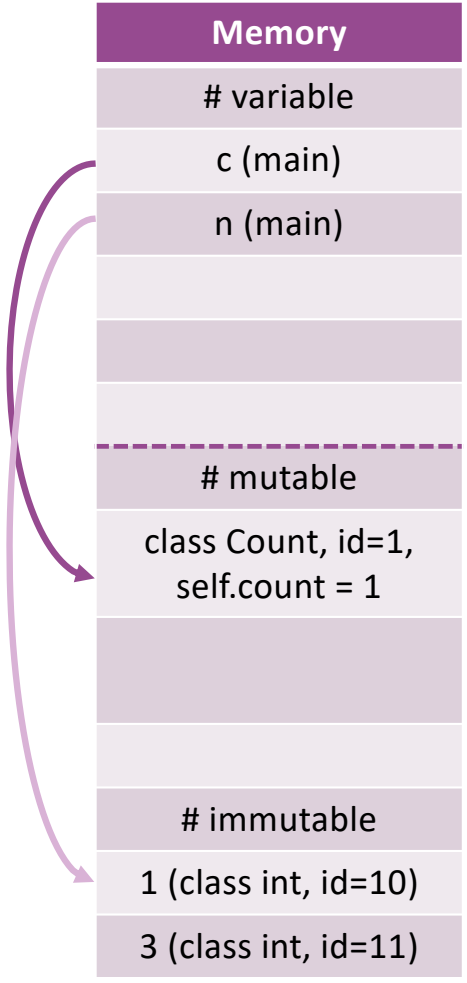
```
class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    n = 1
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c = Count(5)
    n = 3

main() # Call the main function
```



# Diving into mutable/immutable objects (I)

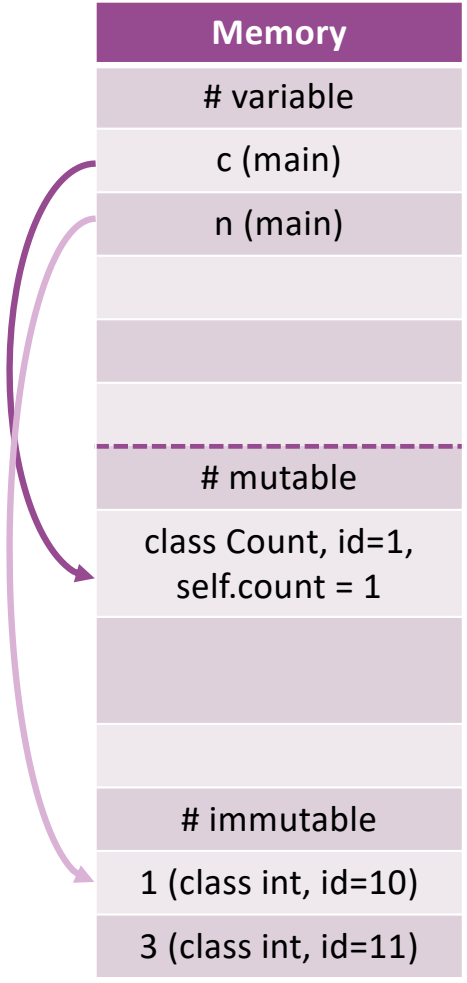
```
class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    n = 1
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c = Count(5)
    n = 3

main() # Call the main function
```



# Diving into mutable/immutable objects (II)

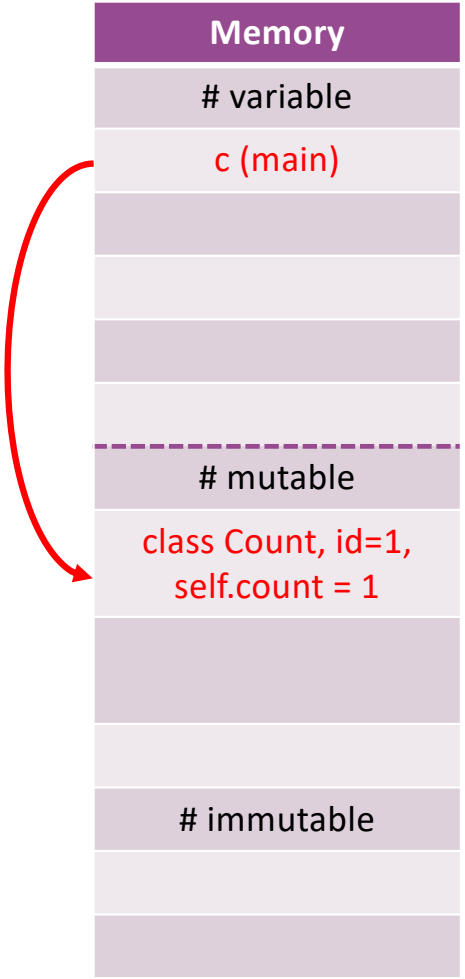
```
class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    n = 1
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c.count = 5
    n = 3

main() # Call the main function
```



# Diving into mutable/immutable objects (II)

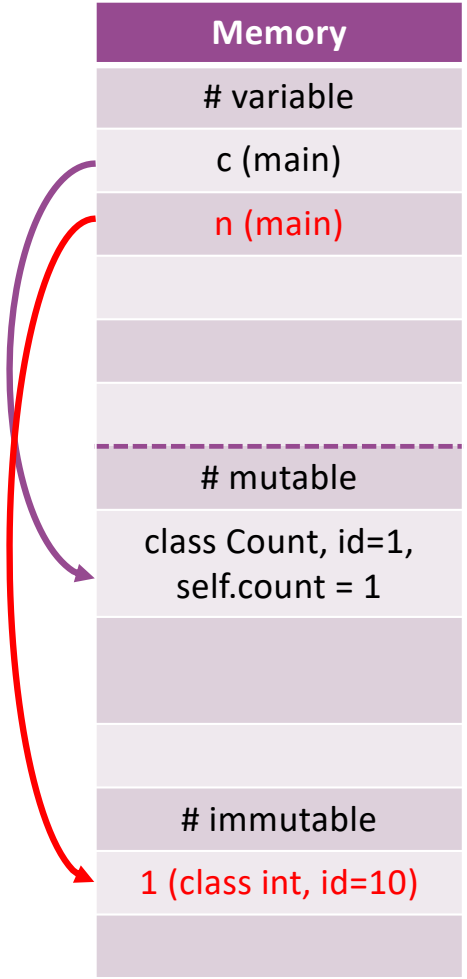
```
class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    n = 1
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c.count = 5
    n = 3

main() # Call the main function
```



# Diving into mutable/immutable objects (II)

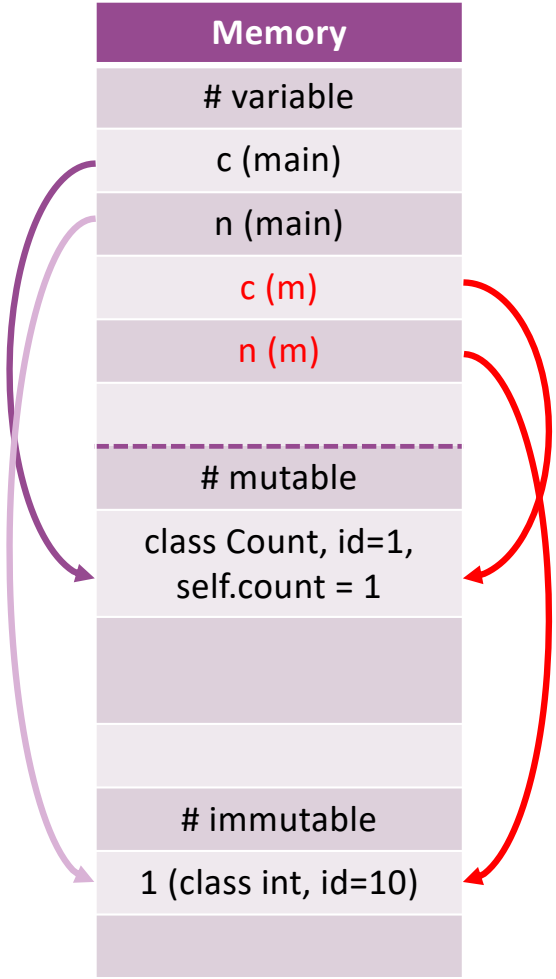
```
class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    n = 1
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c.count = 5
    n = 3

main() # Call the main function
```



# Diving into mutable/immutable objects (II)

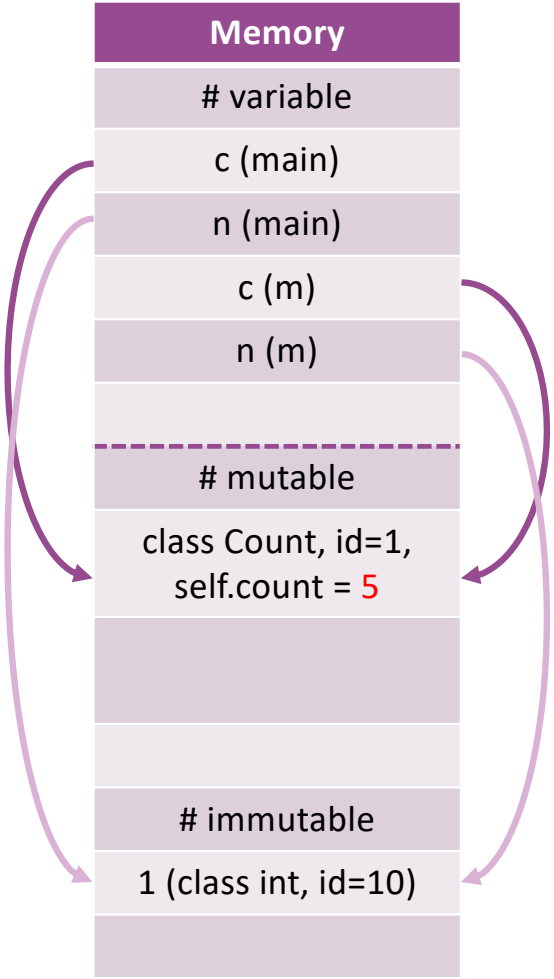
```
class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    n = 1
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c.count = 5
    n = 3

main() # Call the main function
```



# Diving into mutable/immutable objects (II)

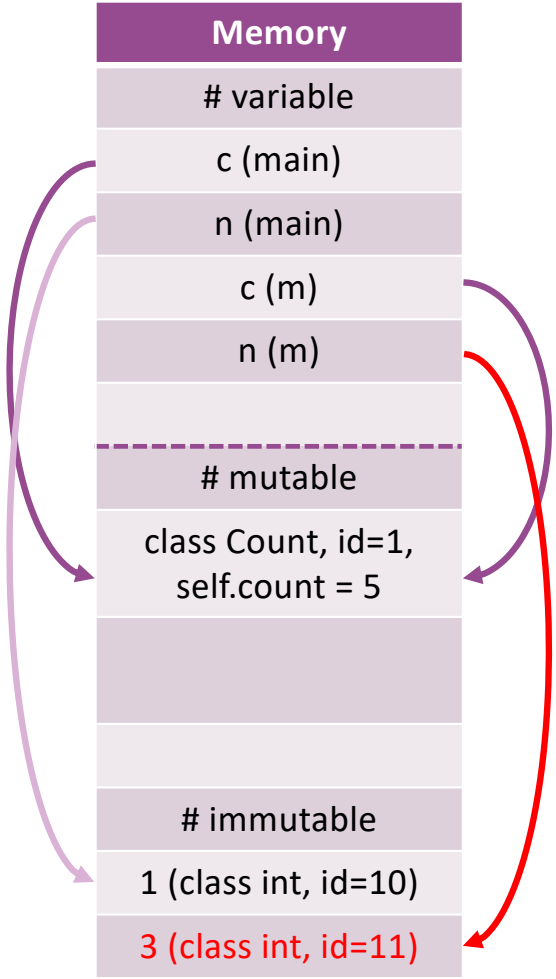
```
class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    n = 1
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c.count = 5
    n = 3

main() # Call the main function
```



# Diving into mutable/immutable objects (II)

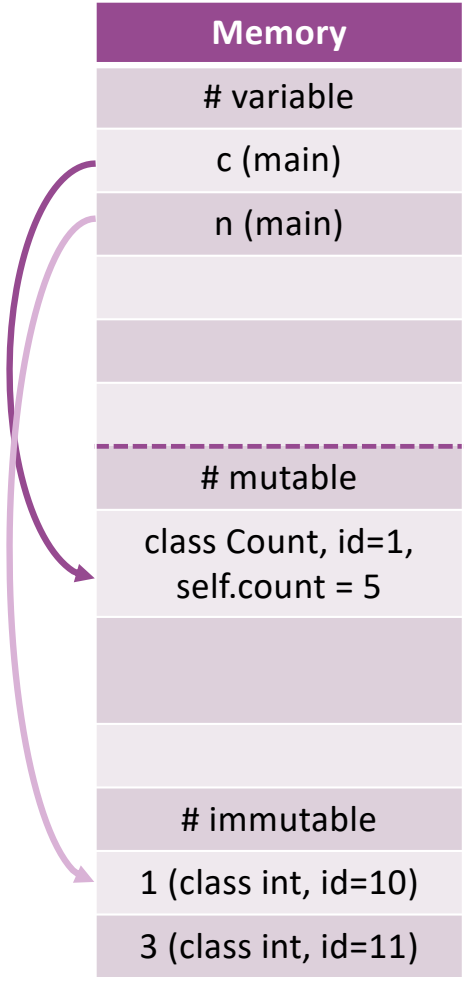
```
class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    n = 1
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c.count = 5
    n = 3

main() # Call the main function
```



# Diving into mutable/immutable objects (II)

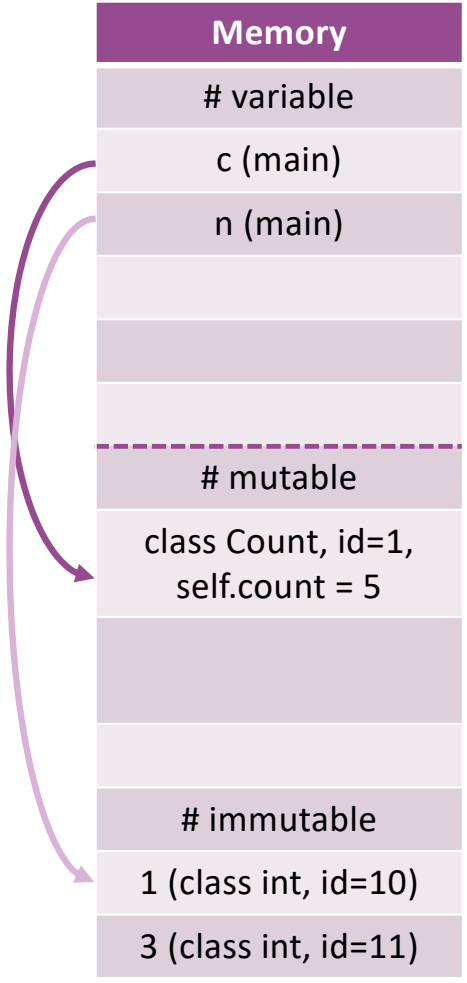
```
class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    n = 1
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c.count = 5
    n = 3

main() # Call the main function
```



# Diving into mutable/immutable objects (II)

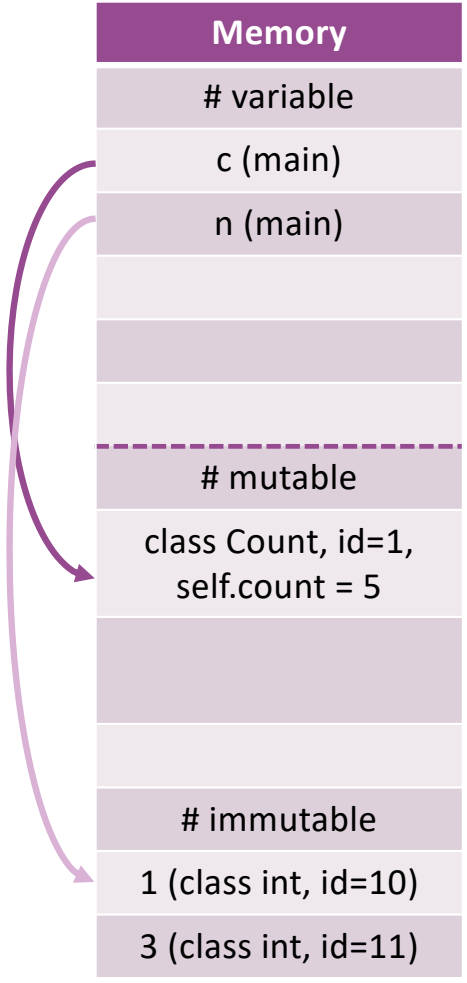
```
class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    n = 1
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c.count = 5
    n = 3

main() # Call the main function
```



# Revisit *is* operator

- The *is* operator checks whether two variables are references to the same object address

- Difference between mutable & immutable objects
  - Immutable objects do not change throughout the lifecycle of a program, so we don't need to create multiple copies of the same value. All variables with the same value reference the same memory address
  - Mutable objects' data fields may change during the program execution, so whenever a new object is created, a new memory space is allocated. Each mutable objects are different, even though they might have the same value during some periods

| Memory |
| --- |
| # mutable |
| class Count, id=1, self.count = 5 |
| |
| |
| # immutable |
| 1 (class int, id=10) |
| 3 (class int, id=11) |

# Practice

- Verify the above contents using id()

Thanks