Lab 9 GRS

recursion, call stack

Recursion

- Solving a problem using the same problem
  - In computer science, it refers to **calling a function on itself**
- Recursion requires a base case and a recursive case
  - Base case: recursive case will lead to this, stops recursion
  - Recursive case: function call that leads to base case

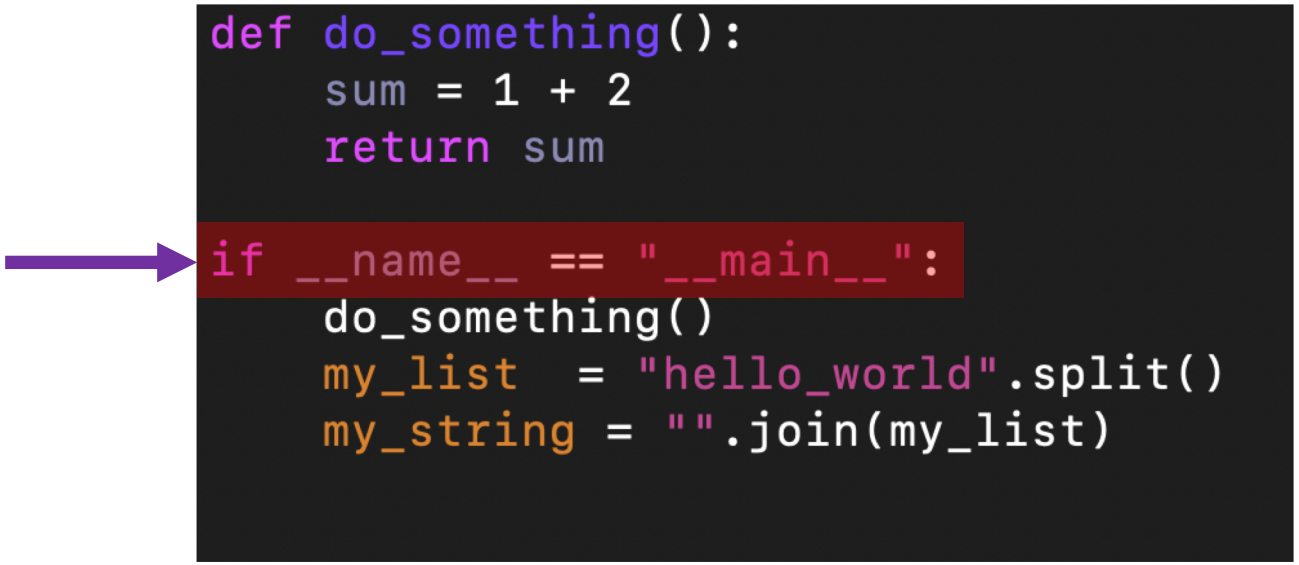
Recursion

- What's the sum from 1-1000?
  - I don't the sum from 1-1000, but I **do** know it's 1 + the sum from 2-1000
  - Base case?
  - Recursive case?
- PEMDAS
- Chain rule

How (and why) does recursion work?

- The call stack
  - The call stack is a **stack** data structure
  - Stack follows LIFO (**L**ast **I**n, **F**irst **O**ut)
  - The call stack keeps track of **what** point of the program we are in, as well what **data we can currently access**
    - The top of the stack is the current function call/point in the program
    - This is why we can **only** access local (and global) variables
    - When we try accessing data out of our scope, it's likely somewhere below
  - When program leaves a function, the function call is **popped off the stack**

A ‘normal’ program running and its call stack



```
def do_something():
    sum = 1 + 2
    return sum

if __name__ == "__main__":
    do_something()
    my_list  = "hello_world".split()
    my_string = "".join(my_list)
```

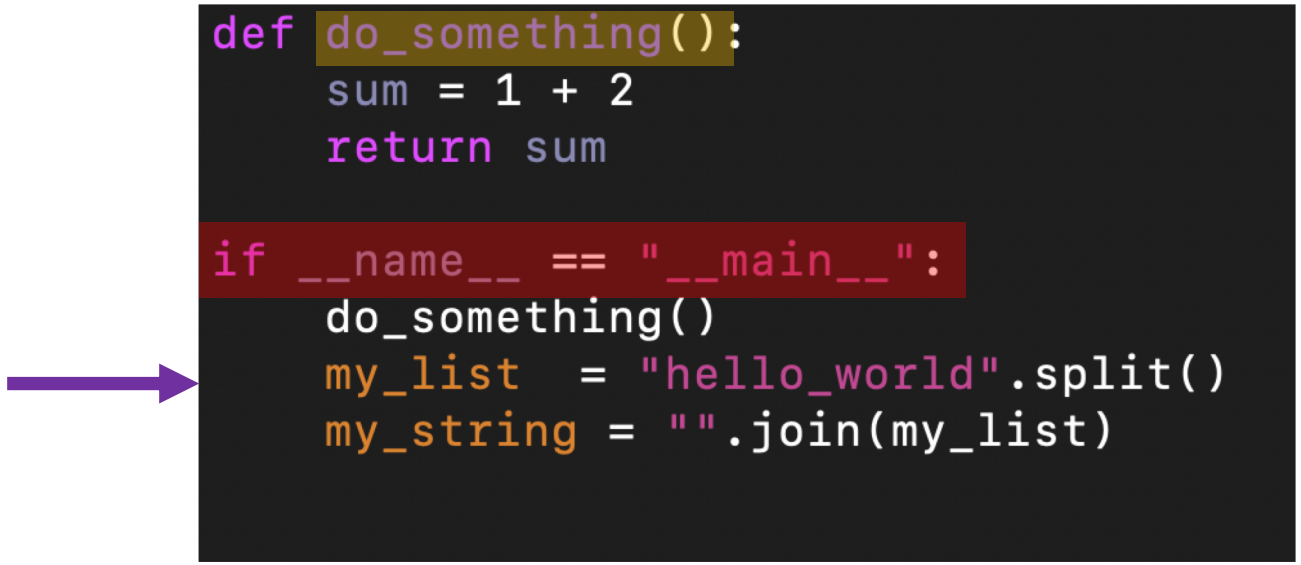
Call stack

A ‘normal’ program running and its call stack

```
def do_something():
    sum = 1 + 2
    return sum

if __name__ == "__main__":
    do_something()
    my_list  = "hello_world".split()
    my_string = "".join(my_list)
```

Call stack

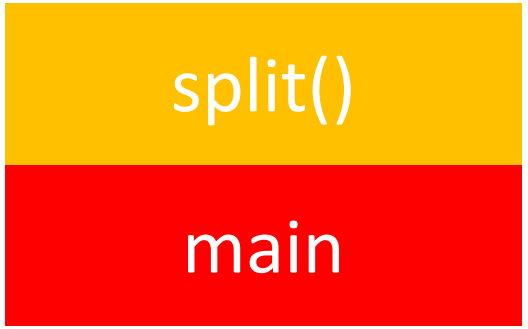
A ‘normal’ program running and its call stack



Call stack

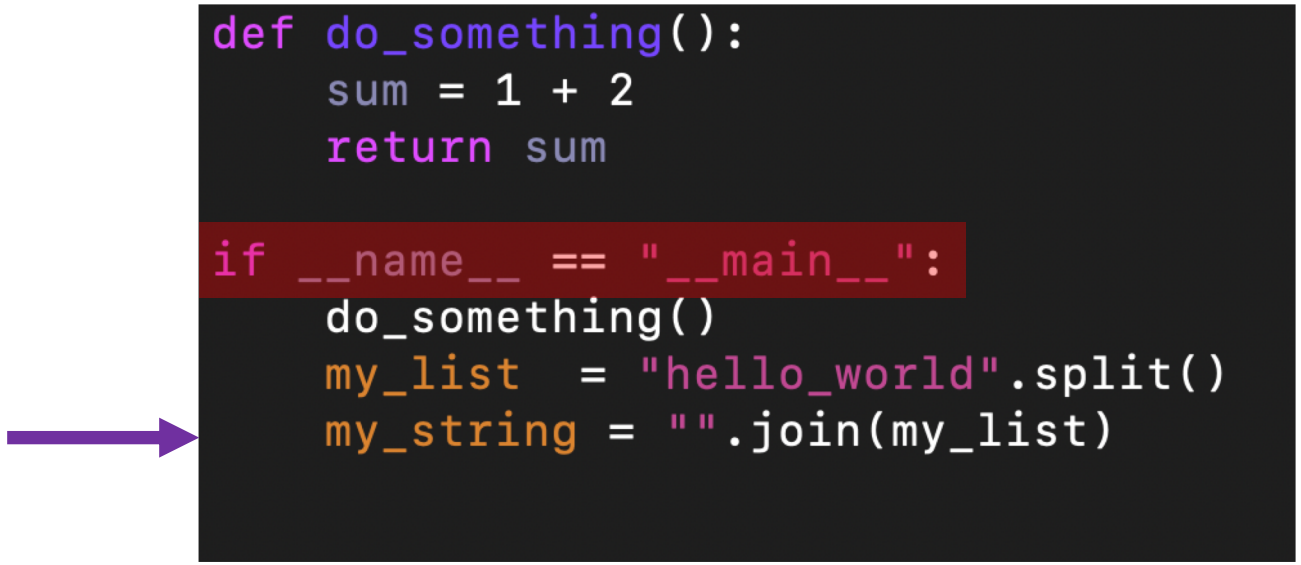
A ‘normal’ program running and its call stack

```
def do_something():
    sum = 1 + 2
    return sum

if __name__ == "__main__":
    do_something()
    my_list  = "hello_world".split()
    my_string = "".join(my_list)
```



Call stack

A ‘normal’ program running and its call stack



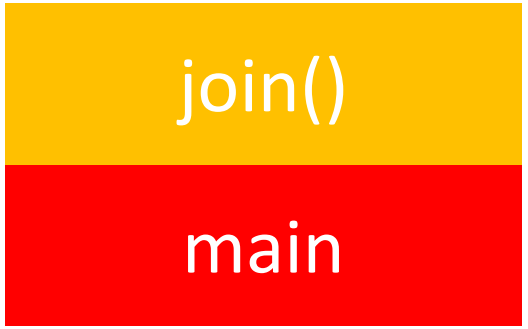
```
def do_something():
    sum = 1 + 2
    return sum

if __name__ == "__main__":
    do_something()
    my_list  = "hello_world".split()
    my_string = "".join(my_list)
```

main

Call stack

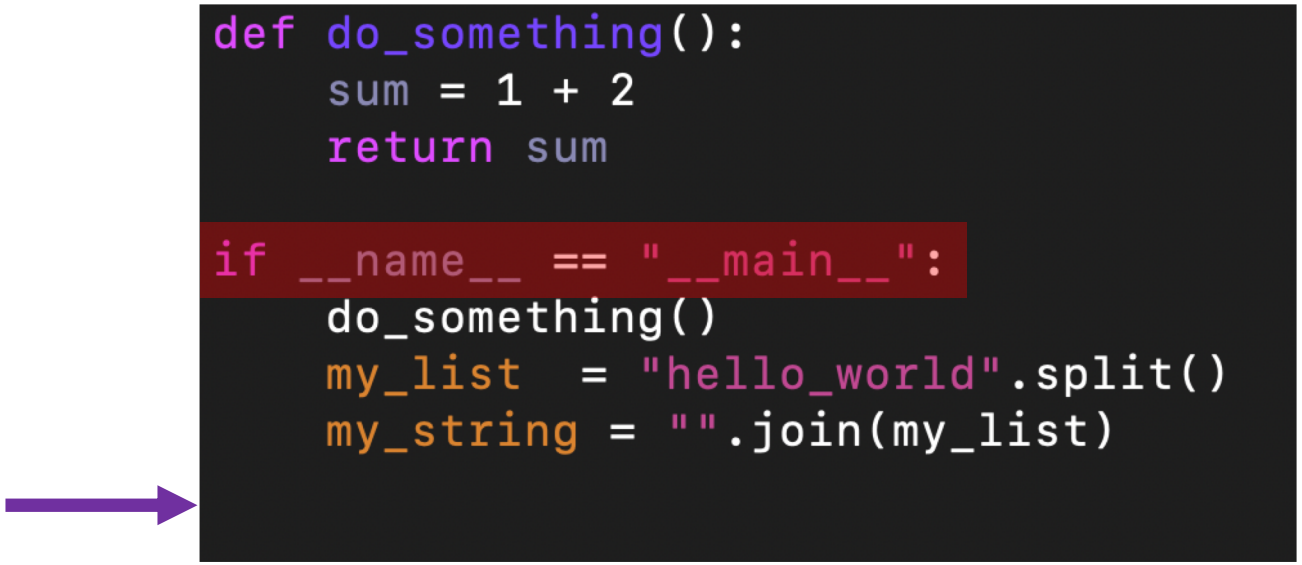
A ‘normal’ program running and its call stack

```
def do_something():
    sum = 1 + 2
    return sum

if __name__ == "__main__":
    do_something()
    my_list  = "hello_world".split()
    my_string = "".join(my_list)
```



Call stack

A ‘normal’ program running and its call stack



```
def do_something():
    sum = 1 + 2
    return sum

if __name__ == "__main__":
    do_something()
    my_list  = "hello_world".split()
    my_string = "".join(my_list)
```

Call stack

A ‘normal’ program running and its call stack

```
def do_something():
    sum = 1 + 2
    return sum

if __name__ == "__main__":
    do_something()
    my_list  = "hello_world".split()
    my_string = "".join(my_list)
```

Call stack
-it’s **empty**, so we’re done and the program has ran successfully

A call stack of a program using recursion

```
def recursive_sum(num):
    #base case
    if num == 0:
        return 0

    #recursive case
    return num + recursive_sum(num-1)

if __name__ == "__main__":
    num = 2
    my_sum = recursive_sum(num)
```

Call stack

A call stack of a program using recursion

```
def recursive_sum(num):
    #base case
    if num == 0:
        return 0

    #recursive case
    return num + recursive_sum(num-1)

if __name__ == "__main__":
    num = 2
    my_sum = recursive_sum(num)
```

Call stack

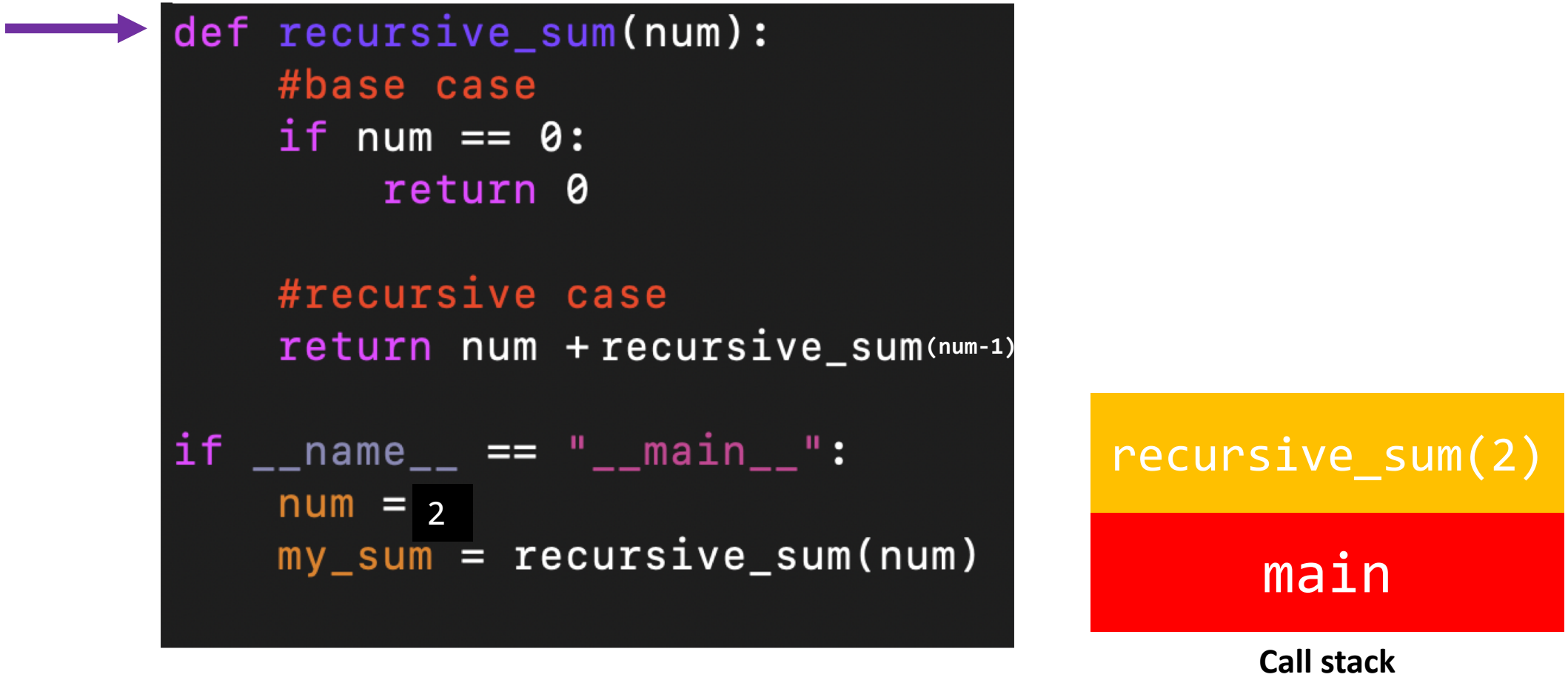
A call stack of a program using recursion

```
def recursive_sum(num):
    #base case
    if num == 0:
        return 0

    #recursive case
    return num + recursive_sum(num-1)

if __name__ == "__main__":
    num = 2
    my_sum = recursive_sum(num)
```

Call stack

A call stack of a program using recursion

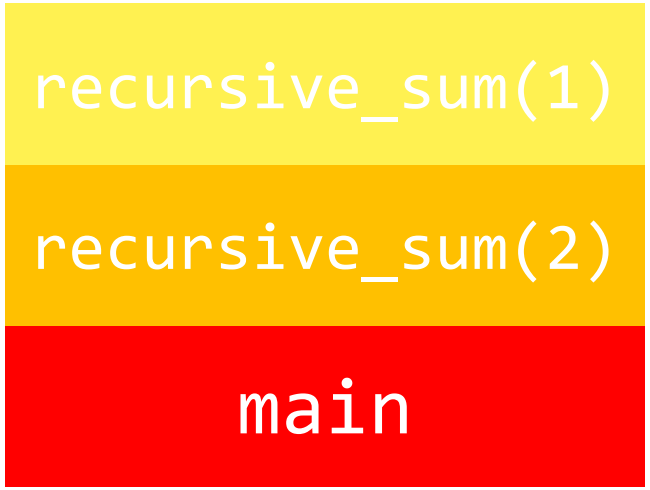
```
def recursive_sum(num):
    #base case
    if num == 0:
        return 0

    #recursive case
    return num + recursive_sum(num-1)

if __name__ == "__main__":
    num = 2
    my_sum = recursive_sum(num)
```



Call stack

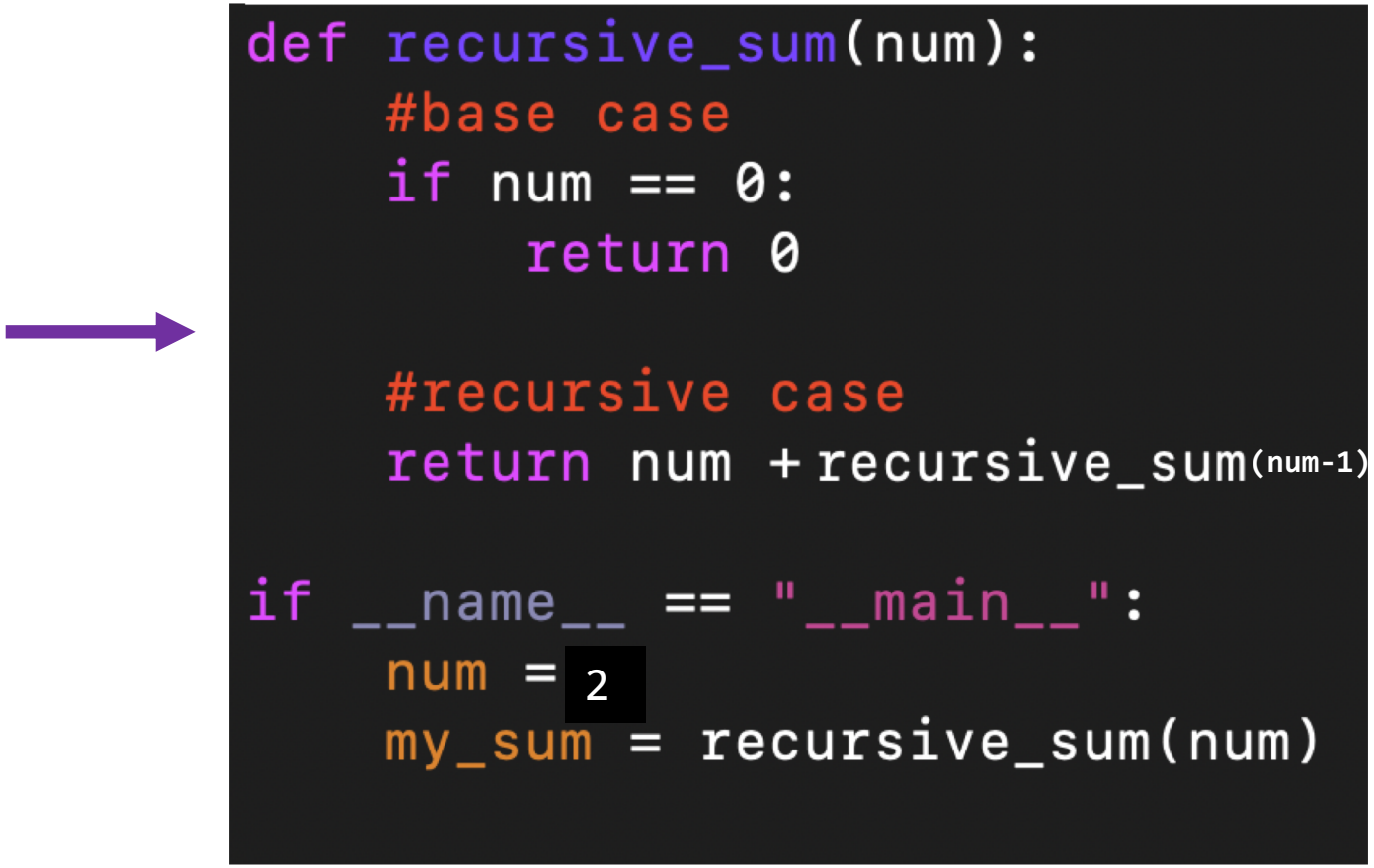
A call stack of a program using recursion

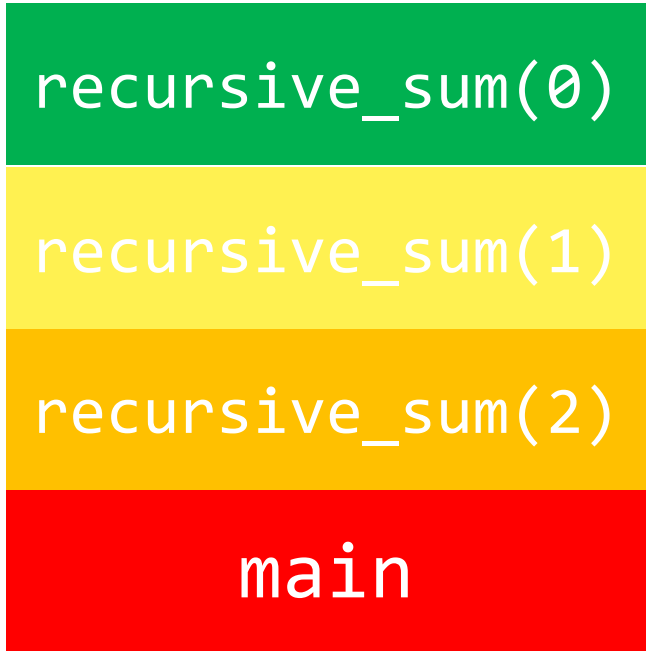
```
def recursive_sum(num):
    #base case
    if num == 0:
        return 0

    #recursive case
    return num + recursive_sum(num-1)

if __name__ == "__main__":
    num = 2
    my_sum = recursive_sum(num)
```



Call stack

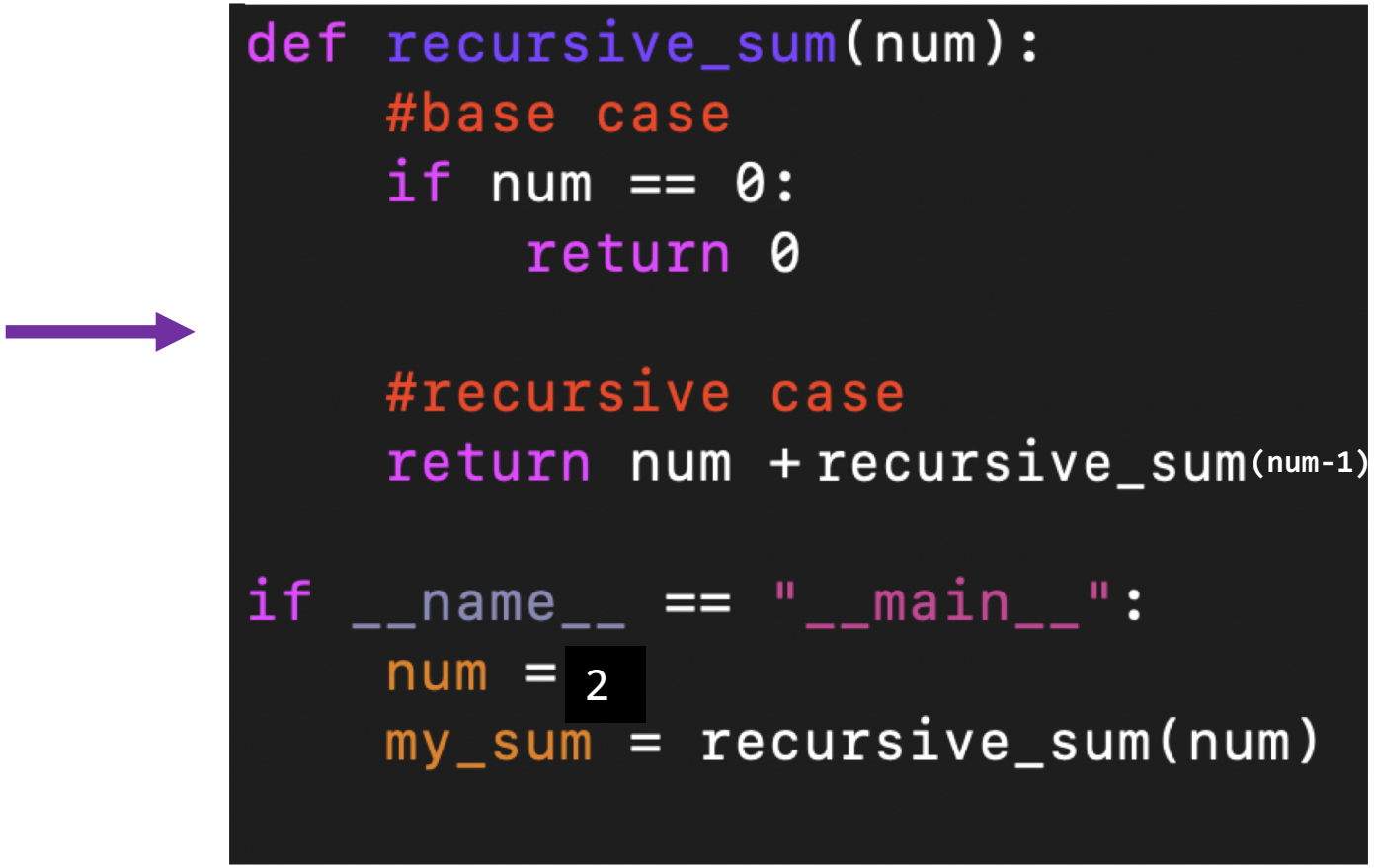
A call stack of a program using recursion

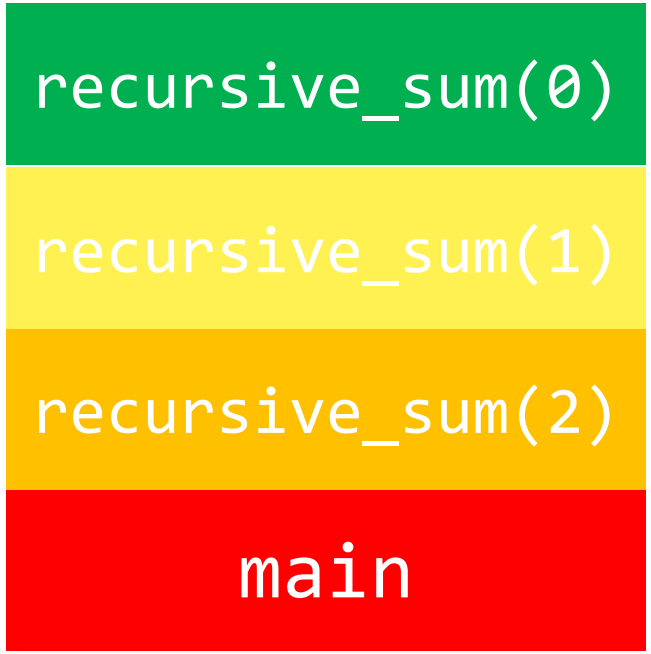




Call stack

A call stack of a program using recursion

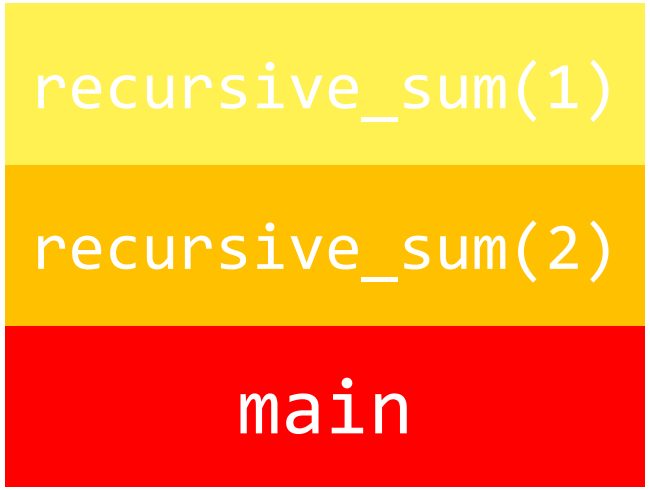




Call stack

A call stack of a program using recursion

```
def recursive_sum(num):
    #base case
    if num == 0:
        return 0

    #recursive case
    return num + recursive_sum(num-1)

if __name__ == "__main__":
    num = 2
    my_sum = recursive_sum(num)
```



Call stack

A call stack of a program using recursion

```
def recursive_sum(num):
    #base case
    if num == 0:
        return 0

    #recursive case
    return num + recursive_sum(num-1)

if __name__ == "__main__":
    num = 2
    my_sum = recursive_sum(num)
```

Call stack

A call stack of a program using recursion

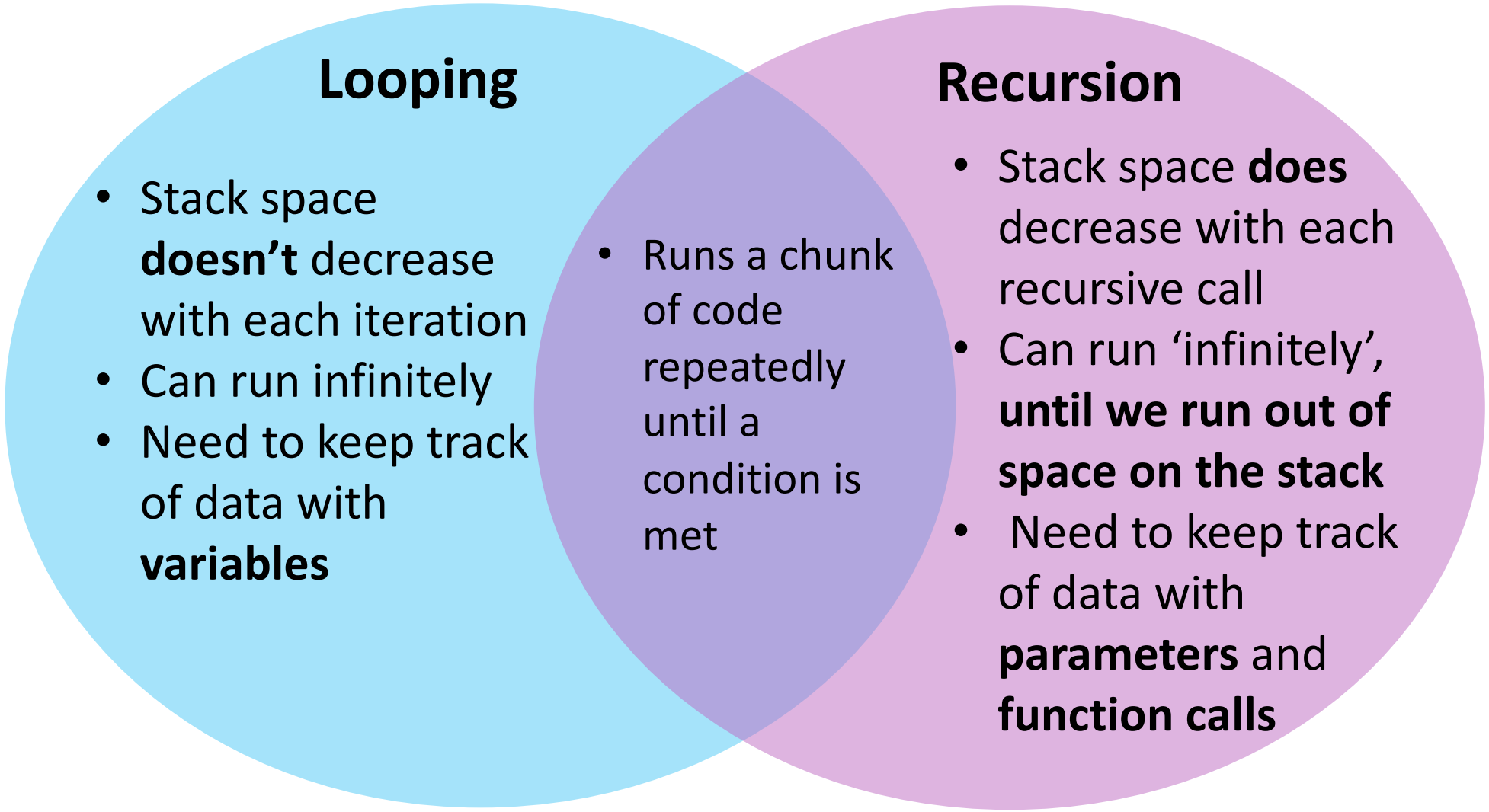
```
def recursive_sum(num):
    #base case
    if num == 0:
        return 0

    #recursive case
    return num + recursive_sum(num-1)

if __name__ == "__main__":
    num = 2
    my_sum = recursive_sum(num)
```

Call stack

Recursion examples

- Factorial
- Fibonacci numbers
- Pascal’s triangle

Differences between looping and recursion



GRS Activity

- cp /afs/umbc.edu/users/a/g/agatha3/pub/201_grs/lab9/activity.py .