# Introduction to Programming in Python

Procedural Programming: Recursion

## Outline

# Recursive Functions

## Recursive Functions

A recursive function is one that

- Calls itself
- Has a base case
- Addresses smaller, non overlapping subproblems in each recursive call

$$n! = \begin{cases} n(n-1)! & \text{if } n > 0, \text{ and} \\ 1 & \text{if } n = 0 \end{cases}$$

## **Examples** · Factorial Function

$$n! = \begin{cases} n(n-1)! & \text{if } n > 0, \text{ and} \\ 1 & \text{if } n = 0 \end{cases}$$

```
def _factorial(n):
    if n == 0:
        return 1
    return n * _factorial(n - 1)
```

$$n! = \begin{cases} n(n-1)! & \text{if } n > 0, \text{ and} \\ 1 & \text{if } n = 0 \end{cases}$$

```
def _factorial(n):
    if n == 0:
        return 1
    return n * _factorial(n - 1)
```

Call trace for `_factorial(5)`

```
x = _factorial(5)
```

$$n! = \begin{cases} n(n-1)! & \text{if } n > 0, \text{ and} \\ 1 & \text{if } n = 0 \end{cases}$$

```
def _factorial(n):
    if n == 0:
        return 1
    return n * _factorial(n - 1)
```

Call trace for `_factorial(5)`

```
x = _factorial(5)
          return 5 * _factorial(4)
```

$$n! = \begin{cases} n(n-1)! & \text{if } n > 0, \text{ and} \\ 1 & \text{if } n = 0 \end{cases}$$

```
def _factorial(n):
    if n == 0:
        return 1
    return n * _factorial(n - 1)
```

Call trace for `_factorial(5)`

```
x = _factorial(5)
          return 5 * _factorial(4)
                        return 4 * _factorial(3)
```

$$n! = \begin{cases} n(n-1)! & \text{if } n > 0, \text{ and} \\ 1 & \text{if } n = 0 \end{cases}$$

```
def _factorial(n):
    if n == 0:
        return 1
    return n * _factorial(n - 1)
```

Call trace for `_factorial(5)`

```
x = _factorial(5)
          return 5 * _factorial(4)
                        return 4 * _factorial(3)
                                      return 3 * _factorial(2)
```

$$n! = \begin{cases} n(n-1)! & \text{if } n > 0, \text{ and} \\ 1 & \text{if } n = 0 \end{cases}$$

```
def _factorial(n):
    if n == 0:
        return 1
    return n * _factorial(n - 1)
```

Call trace for `_factorial(5)`

```
x = _factorial(5)
          return 5 * _factorial(4)
                        return 4 * _factorial(3)
                                      return 3 * _factorial(2)
                                                    return 2 * _factorial(1)
```

$$n! = \begin{cases} n(n-1)! & \text{if } n > 0, \text{ and} \\ 1 & \text{if } n = 0 \end{cases}$$

```
def _factorial(n):
    if n == 0:
        return 1
    return n * _factorial(n - 1)
```

Call trace for `_factorial(5)`

```
x = _factorial(5)
          return 5 * _factorial(4)
                        return 4 * _factorial(3)
                                      return 3 * _factorial(2)
                                                    return 2 * _factorial(1)
                                                                  return 1 * _factorial(0)
```

$$n! = \begin{cases} n(n-1)! & \text{if } n > 0, \text{ and} \\ 1 & \text{if } n = 0 \end{cases}$$

```
def _factorial(n):
    if n == 0:
        return 1
    return n * _factorial(n - 1)
```

Call trace for `_factorial(5)`

```
x = _factorial(5)
          return 5 * _factorial(4)
                     return 4 * _factorial(3)
                                return 3 * _factorial(2)
                                           return 2 * _factorial(1)
                                                      return 1 * _factorial(0)
                                                                 return 1
```

$$n! = \begin{cases} n(n-1)! & \text{if } n > 0, \text{ and} \\ 1 & \text{if } n = 0 \end{cases}$$

```
def _factorial(n):
    if n == 0:
        return 1
    return n * _factorial(n - 1)
```

Call trace for `_factorial(5)`

```
x = _factorial(5)
          return 5 * _factorial(4)
                        return 4 * _factorial(3)
                                      return 3 * _factorial(2)
                                                    return 2 * _factorial(1)
                                                                  return 1 * 1
```

$$n! = \begin{cases} n(n-1)! & \text{if } n > 0, \text{ and} \\ 1 & \text{if } n = 0 \end{cases}$$

```
def _factorial(n):
    if n == 0:
        return 1
    return n * _factorial(n - 1)
```

Call trace for `_factorial(5)`

```
x = _factorial(5)
          return 5 * _factorial(4)
                        return 4 * _factorial(3)
                                      return 3 * _factorial(2)
                                                    return 2 * 1
```

$$n! = \begin{cases} n(n-1)! & \text{if } n > 0, \text{ and} \\ 1 & \text{if } n = 0 \end{cases}$$

```
def _factorial(n):
    if n == 0:
        return 1
    return n * _factorial(n - 1)
```

Call trace for `_factorial(5)`

```
x = _factorial(5)
          return 5 * _factorial(4)
                        return 4 * _factorial(3)
                                      return 3 * 2
```

$$n! = \begin{cases} n(n-1)! & \text{if } n > 0, \text{ and} \\ 1 & \text{if } n = 0 \end{cases}$$

```
def _factorial(n):
    if n == 0:
        return 1
    return n * _factorial(n - 1)
```

Call trace for `_factorial(5)`

```
x = _factorial(5)
          return 5 * _factorial(4)
                        return 4 * 6
```

$$n! = \begin{cases} n(n-1)! & \text{if } n > 0, \text{ and} \\ 1 & \text{if } n = 0 \end{cases}$$

```
def _factorial(n):
    if n == 0:
        return 1
    return n * _factorial(n - 1)
```

Call trace for `_factorial(5)`

```
x = _factorial(5)
          return 5 * 24
```

$$n! = \begin{cases} n(n-1)! & \text{if } n > 0, \text{ and} \\ 1 & \text{if } n = 0 \end{cases}$$

```
def _factorial(n):
    if n == 0:
        return 1
    return n * _factorial(n - 1)
```

Call trace for `_factorial(5)`

```
x = 120
```

factorial.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | $n!$ |

factorial.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | $n!$ |

>_ ~/workspace/ipp/programs

```
$ _
```

factorial.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | $n!$ |

~/workspace/ipp/programs

```
$ python3 factorial.py 0
```

factorial.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | $n!$ |

~/workspace/ipp/programs

```
$ python3 factorial.py 0
1
$ _
```

factorial.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | $n!$ |

~/workspace/ipp/programs

```
$ python3 factorial.py 0
1
$ python3 factorial.py 5
```

factorial.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | $n!$ |

~/workspace/ipp/programs

```
$ python3 factorial.py 0
1
$ python3 factorial.py 5
120
$ _
```

## **Examples** · Factorial Function

</> factorial.py

```python
import stdio
import sys

def main():
    n = int(sys.argv[1])
    stdio.writeln(_factorial(n))

def _factorial(n):
    if n == 0:
        return 1
    return n * _factorial(n - 1)

if __name__ == "__main__":
    main()
```

$$\gcd(p, q) = \begin{cases} \gcd(q, p \bmod q) & \text{if } q \neq 0, \text{ and} \\ p & \text{if } q = 0 \end{cases}$$

$$\gcd(p, q) = \begin{cases} \gcd(q, p \bmod q) & \text{if } q \neq 0, \text{ and} \\ p & \text{if } q = 0 \end{cases}$$

```
def _gcd(p, q):
    if q == 0:
        return p
    return _gcd(q, p % q)
```

$$\gcd(p, q) = \begin{cases} \gcd(q, p \bmod q) & \text{if } q \neq 0, \text{ and} \\ p & \text{if } q = 0 \end{cases}$$

```
def _gcd(p, q):
    if q == 0:
        return p
    return _gcd(q, p % q)
```

Call trace for `_gcd(1440, 408)`

```
_gcd(1440, 408)
  _gcd(408, 216)
    _gcd(216, 192)
      _gcd(192, 24)
        _gcd(24, 0)
          return 24
        return 24
      return 24
    return 24
  return 24
```

gcd.py

| | |
|---|---|
| Command-line input | $p$ (int) and $q$ (int) |
| Standard output | $\gcd(p, q)$ |

gcd.py

| | |
|---|---|
| Command-line input | $p$ (int) and $q$ (int) |
| Standard output | $\gcd(p, q)$ |

>_ ~/workspace/ipp/programs

```
$ _
```

gcd.py

| | |
|---|---|
| Command-line input | $p$ (int) and $q$ (int) |
| Standard output | $\gcd(p, q)$ |

~/workspace/ipp/programs

```
$ python3 gcd.py 1440 408
```

gcd.py

| | |
|---|---|
| Command-line input | $p$ (int) and $q$ (int) |
| Standard output | $\gcd(p, q)$ |

~/workspace/ipp/programs

```
$ python3 gcd.py 1440 408
24
$ _
```

gcd.py

| | |
|---|---|
| Command-line input | $p$ (int) and $q$ (int) |
| Standard output | $\gcd(p, q)$ |

~/workspace/ipp/programs

```
$ python3 gcd.py 1440 408
24
$ python3 314159 271828
```

gcd.py

| | |
|---|---|
| Command-line input | $p$ (int) and $q$ (int) |
| Standard output | $\gcd(p, q)$ |

~/workspace/ipp/programs

```
$ python3 gcd.py 1440 408
24
$ python3 314159 271828
1
$ _
```

# **Examples** · GCD Function

</> gcd.py

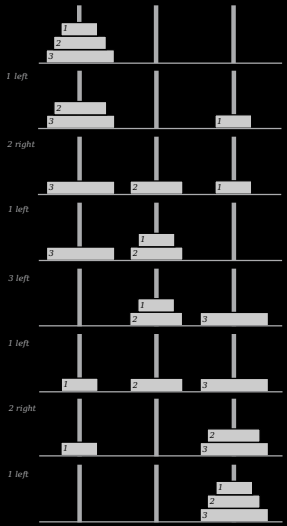
```
import stdio
import sys

def main():
    p = int(sys.argv[1])
    q = int(sys.argv[2])
    stdio.writeln(_gcd(p, q))

def _gcd(p, q):
    if q == 0:
        return p
    return _gcd(q, p % q)

if __name__ == "__main__":
    main()
```

# **Examples** · Towers of Hanoi



towersofhanoi.py

| | |
|---|---|
| Command-line input | *n* (int) |
| Standard output | instructions to move n Towers of Hanoi disks to the left |

towersofhanoi.py

| | |
|---|---|
| Command-line input | *n* (int) |
| Standard output | instructions to move n Towers of Hanoi disks to the left |

>_ ~/workspace/ipp/programs

```
$ _
```

towersofhanoi.py

| | |
|---|---|
| Command-line input | *n* (int) |
| Standard output | instructions to move n Towers of Hanoi disks to the left |

>_ ~/workspace/ipp/programs

```
$ python3 towersofhanoi.py 1
```

towersofhanoi.py

| | |
|---|---|
| Command-line input | *n* (int) |
| Standard output | instructions to move n Towers of Hanoi disks to the left |

>_ ~/workspace/ipp/programs

```
$ python3 towersofhanoi.py 1
1 left
$ _
```

towersofhanoi.py

| | |
|---|---|
| Command-line input | *n* (int) |
| Standard output | instructions to move n Towers of Hanoi disks to the left |

>_ ~/workspace/ipp/programs

```
$ python3 towersofhanoi.py 1
1 left
$ python3 towersofhanoi.py 2
```

towersofhanoi.py

| | |
|---|---|
| Command-line input | *n* (int) |
| Standard output | instructions to move n Towers of Hanoi disks to the left |

>_ ~/workspace/ipp/programs

```
$ python3 towersofhanoi.py 1
1 left
$ python3 towersofhanoi.py 2
1 right
2 left
1 right
$ _
```

towersofhanoi.py

| | |
|---|---|
| Command-line input | *n* (int) |
| Standard output | instructions to move n Towers of Hanoi disks to the left |

~/workspace/ipp/programs

```
$ python3 towersofhanoi.py 1
1 left
$ python3 towersofhanoi.py 2
1 right
2 left
1 right
$ python3 towersofhanoi.py 3
```

towersofhanoi.py

| | |
|---|---|
| Command-line input | *n* (int) |
| Standard output | instructions to move n Towers of Hanoi disks to the left |

>_ ~/workspace/ipp/programs

```
$ python3 towersofhanoi.py 1
1 left
$ python3 towersofhanoi.py 2
1 right
2 left
1 right
$ python3 towersofhanoi.py 3
1 left
2 right
1 left
3 left
1 left
2 right
1 left
$ _
```

# **Examples** · Towers of Hanoi

towersofhanoi.py

```
import stdio
import sys

def main():
    n = int(sys.argv[1])
    _moves(n, True)

def _moves(n, left):
    if n == 0:
        return
    _moves(n - 1, not left)
    if left:
        stdio.writeln(str(n) + " left")
    else:
        stdio.writeln(str(n) + " right")
    _moves(n - 1, not left)

if __name__ == "__main__":
    main()
```

`htree.py`

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | a level $n$ H-tree centered at $(0.5, 0.5)$ |

htree.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | a level $n$ H-tree centered at $(0.5, 0.5)$ |

```
>_ ~/workspace/ipp/programs
$ _
```

htree.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | a level $n$ H-tree centered at $(0.5, 0.5)$ |

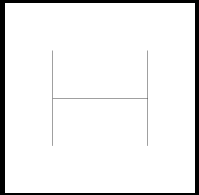
~/workspace/ipp/programs

```
$ python3 htree.py 1
```

htree.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | a level $n$ H-tree centered at $(0.5, 0.5)$ |

```
>_ ~/workspace/ipp/programs
$ python3 htree.py 1
```



htree.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | a level $n$ H-tree centered at $(0.5, 0.5)$ |

~/workspace/ipp/programs

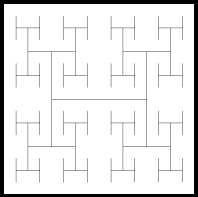
```
$ python3 htree.py 1
$ _
```

```
>_ ~/workspace/ipp/programs
```

```
$ _
```

```
>_ ~/workspace/ipp/programs
$ python3 htree.py 3
```

```
>_ ~/workspace/ipp/programs
$ python3 htree.py 3
```



```
>_ ~/workspace/ipp/programs
```

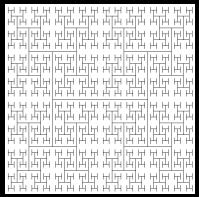
```
$ python3 htree.py 3
$ _
```

```
>_ ~/workspace/ipp/programs
```

```
$ _
```

```
>_ ~/workspace/ipp/programs
$ python3 htree.py 5
```

```
>_ ~/workspace/ipp/programs
$ python3 htree.py 5
```



## Examples · Towers of Hanoi

>_ ~/workspace/ipp/programs

```
$ python3 htree.py 5
$ _
```

</> htree.py

```
import stddraw
import sys

def main():
    n = int(sys.argv[1])
    stddraw.setPenRadius(0.0)
    _draw(n, 0.5, 0.5, 0.5)
    stddraw.show()

def _draw(n, lineLength, x, y):
    if n == 0:
        return
    x0 = x - lineLength / 2
    x1 = x + lineLength / 2
    y0 = y - lineLength / 2
    y1 = y + lineLength / 2
    stddraw.line(x0, y, x1, y)
    stddraw.line(x0, y0, x0, y1)
    stddraw.line(x1, y0, x1, y1)
    _draw(n - 1, lineLength / 2, x0, y0)
    _draw(n - 1, lineLength / 2, x0, y1)
    _draw(n - 1, lineLength / 2, x1, y0)
    _draw(n - 1, lineLength / 2, x1, y1)

if __name__ == "__main__":
    main()
```

$$\text{fib}(n) = \begin{cases} \text{fib}(n-1) + \text{fib}(n-2) & \text{if } n > 1, \text{ and} \\ 1 & \text{if } n = 1, \text{ and} \\ 0 & \text{if } n = 0 \end{cases}$$

$$\text{fib}(n) = \begin{cases} \text{fib}(n-1) + \text{fib}(n-2) & \text{if } n > 1, \text{ and} \\ 1 & \text{if } n = 1, \text{ and} \\ 0 & \text{if } n = 0 \end{cases}$$

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

fibonacci.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | $n$th Fibonacci number |

fibonacci.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | $n$th Fibonacci number |

>_ ~/workspace/ipp/programs

```
$ _
```

fibonacci.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | $n$th Fibonacci number |

>_ ~/workspace/ipp/programs

```
$ python3 fibonacci.py 0
```

fibonacci.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | $n$th Fibonacci number |

~/workspace/ipp/programs

```
$ python3 fibonacci.py 0
0
$ _
```

fibonacci.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | $n$th Fibonacci number |

~/workspace/ipp/programs

```
$ python3 fibonacci.py 0
0
$ python3 fibonacci.py 1
```

fibonacci.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | $n$th Fibonacci number |

~/workspace/ipp/programs

```
$ python3 fibonacci.py 0
0
$ python3 fibonacci.py 1
1
$ _
```

fibonacci.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | $n$th Fibonacci number |

~/workspace/ipp/programs

```
$ python3 fibonacci.py 0
0
$ python3 fibonacci.py 1
1
$ python3 fibonacci.py 2
```

fibonacci.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | $n$th Fibonacci number |

~/workspace/ipp/programs

```
$ python3 fibonacci.py 0
0
$ python3 fibonacci.py 1
1
$ python3 fibonacci.py 2
1
$ _
```

fibonacci.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | $n$th Fibonacci number |

~/workspace/ipp/programs

```
$ python3 fibonacci.py 0
0
$ python3 fibonacci.py 1
1
$ python3 fibonacci.py 2
1
$ python3 fibonacci.py 3
```

fibonacci.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | $n$th Fibonacci number |

~/workspace/ipp/programs

```
$ python3 fibonacci.py 0
0
$ python3 fibonacci.py 1
1
$ python3 fibonacci.py 2
1
$ python3 fibonacci.py 3
2
$ _
```

fibonacci.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | $n$th Fibonacci number |

~/workspace/ipp/programs

```
$ python3 fibonacci.py 0
0
$ python3 fibonacci.py 1
1
$ python3 fibonacci.py 2
1
$ python3 fibonacci.py 3
2
$ python3 fibonacci.py 4
```

fibonacci.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | $n$th Fibonacci number |

~/workspace/ipp/programs

```
$ python3 fibonacci.py 0
0
$ python3 fibonacci.py 1
1
$ python3 fibonacci.py 2
1
$ python3 fibonacci.py 3
2
$ python3 fibonacci.py 4
3
$ _
```

fibonacci.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | $n$th Fibonacci number |

~/workspace/ipp/programs

```
$ python3 fibonacci.py 0
0
$ python3 fibonacci.py 1
1
$ python3 fibonacci.py 2
1
$ python3 fibonacci.py 3
2
$ python3 fibonacci.py 4
3
$ python3 fibonacci.py 10
```

fibonacci.py

| | |
|---|---|
| Command-line input | $n$ (int) |
| Standard output | $n$th Fibonacci number |

~/workspace/ipp/programs

```
$ python3 fibonacci.py 0
0
$ python3 fibonacci.py 1
1
$ python3 fibonacci.py 2
1
$ python3 fibonacci.py 3
2
$ python3 fibonacci.py 4
3
$ python3 fibonacci.py 10
55
$ _
```

## **Examples** · Fibonacci Function

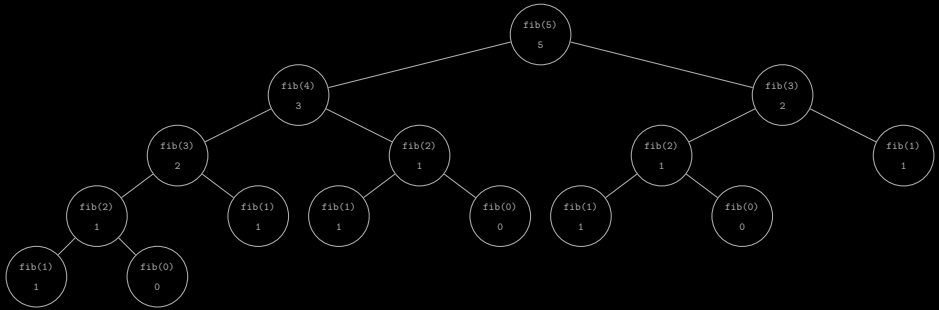
</> fibonacci.py

```
import stdio
import sys

def main():
    n = int(sys.argv[1])
    stdio.writeln(_fibonacci(n))

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

if __name__ == "__main__":
    main()
```


fib(5)
5
fib(4)
3
fib(3)
2
fib(3)
2
fib(2)
1
fib(2)
1
fib(1)
1
fib(2)
1
fib(1)
1
fib(1)
1
fib(0)
0
fib(1)
1
fib(0)
0
fib(1)
1
fib(0)
0
fib(1)
1
fib(0)
0