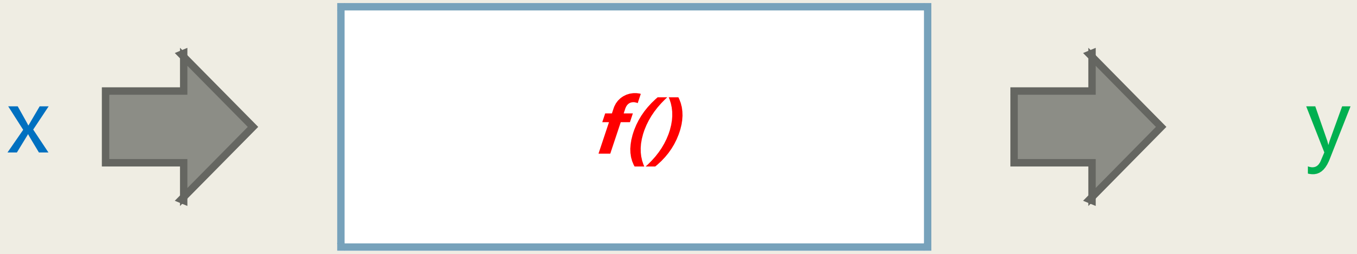
LECTURE 6: FRUITFUL FUNCTIONS

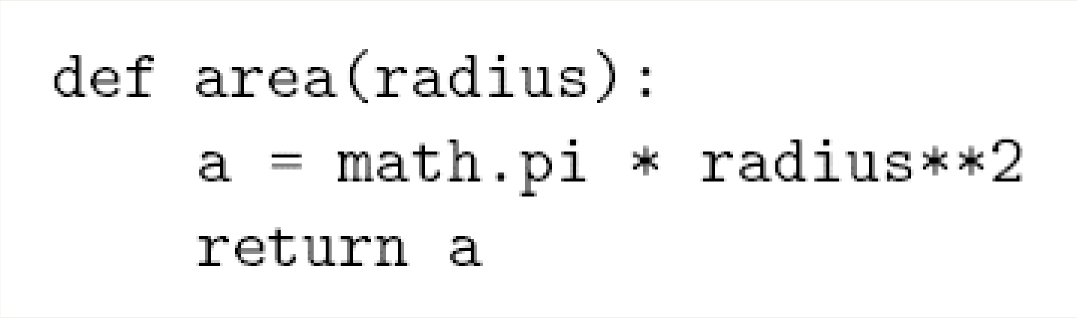
Hung-Yu Wei

Return values in functions

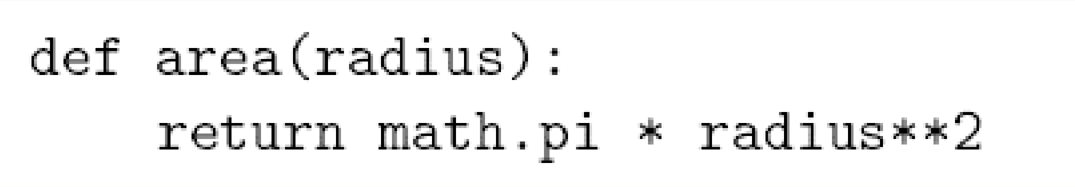
- Review: Functions
  - *Without return*
    - Void functions
  - *With return*
    - fruitful functions



- Example:
  - *function to calculate the **area** of a circle (with radius **r**)*



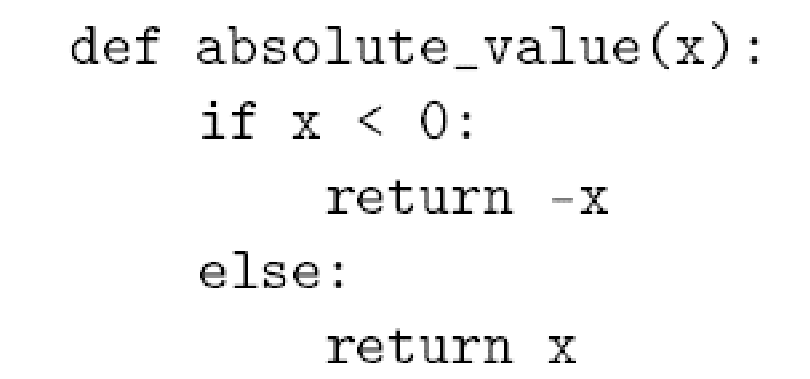
```
def area(radius):
    a = math.pi * radius**2
    return a
```



```
def area(radius):
    return math.pi * radius**2
```

Example: absolute value

- Write your own absolute value function

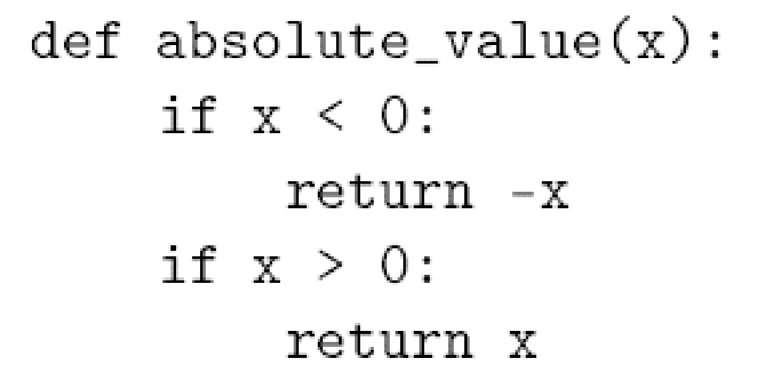


```
def absolute_value(x):
    if x < 0:
        return -x
    else:
        return x
```

- There is built-in function in Python

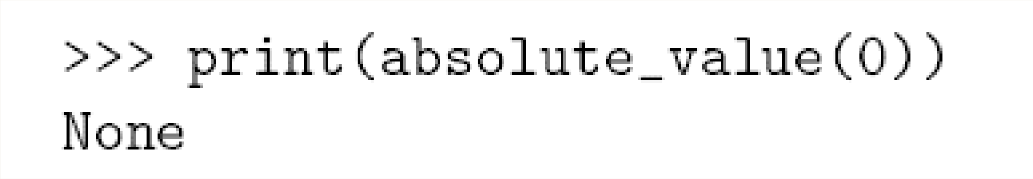
Continued: absolute value

- Make sure there is a return value in all possible cases



```
def absolute_value(x):
    if x < 0:
        return -x
    if x > 0:
        return x
```

- When x = 0 …

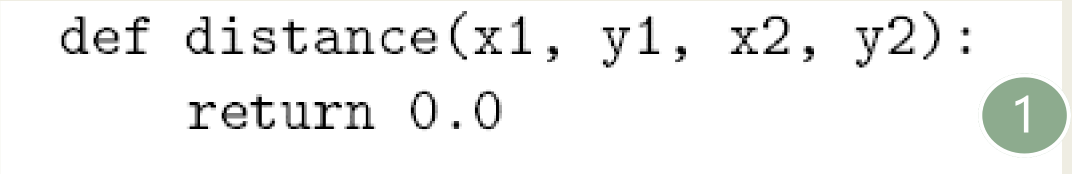


```
>>> print(absolute_value(0))
None
```

Tip: incremental development

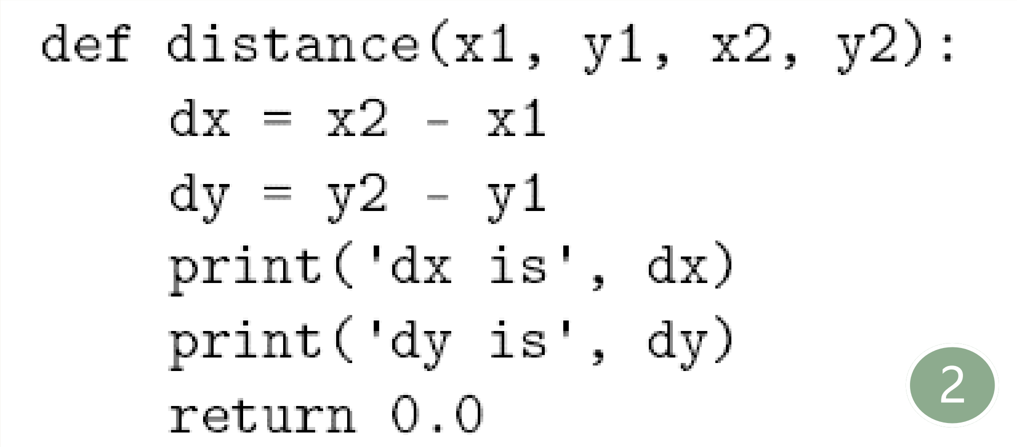
- Avoid long debugging sessions
- Divide a large task into several smaller tasks
- Starting from a smaller block and incrementally expand it
- Steps
  1. *Start with a working program and make small incremental changes. At any point, if there is an error, you should have a good idea where it is.*
  2. *Use variables to hold intermediate values so you can display and check them.*
  3. *Once the program is working, you might want to remove some of the scaffolding or consolidate multiple statements into compound expressions, but only if it does not make the program difficult to read.*

Example: calculating distance



```
def distance(x1, y1, x2, y2):
    return 0.0
```

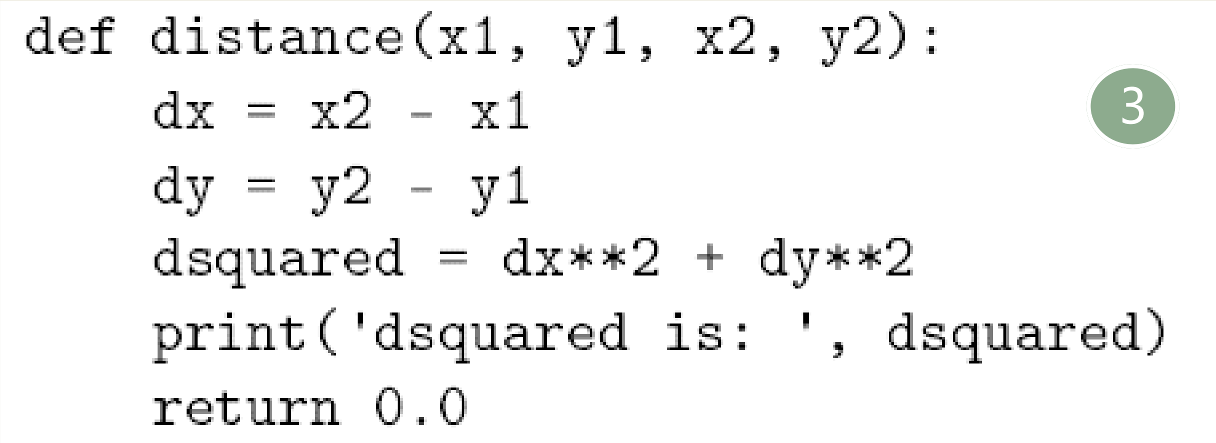
1



```
def distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    print('dx is', dx)
    print('dy is', dy)
    return 0.0
```

2

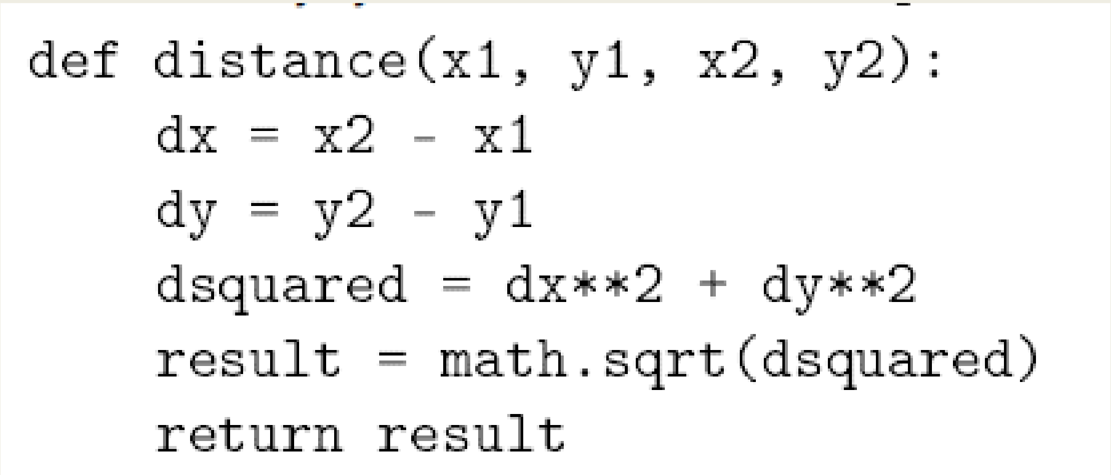
Example:
Calculate distance between (x1,y1) and (x2,y2)



```
def distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    dsquared = dx**2 + dy**2
    print('dsquared is: ', dsquared)
    return 0.0
```

3

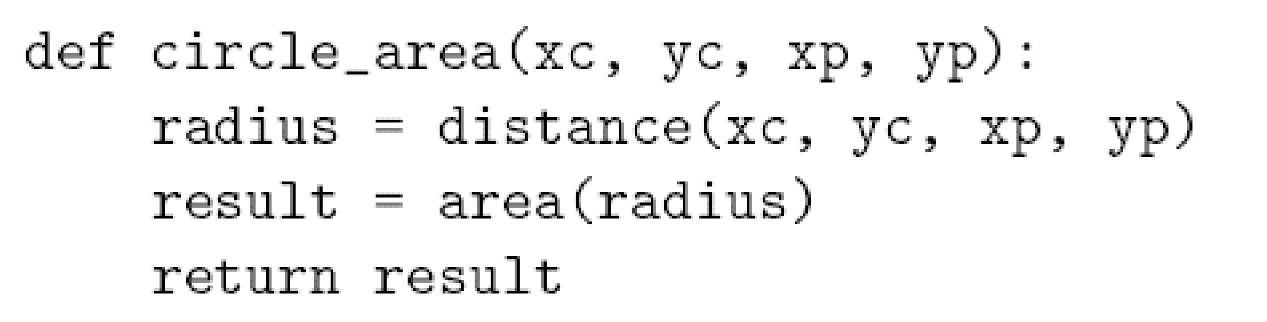
Continued: calculating distance



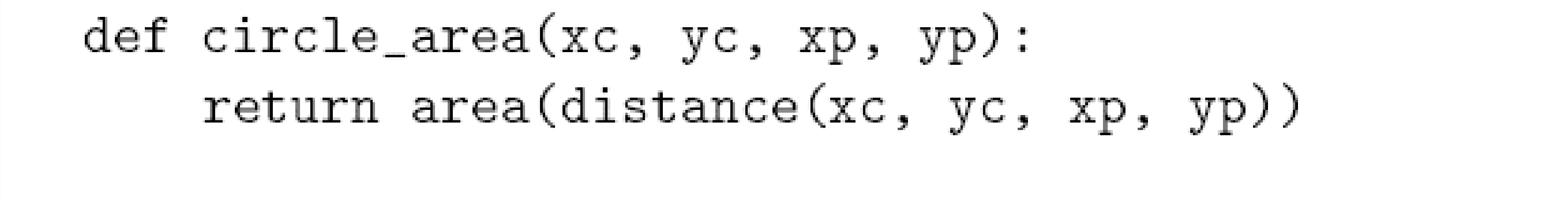
```
def distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    dsquared = dx**2 + dy**2
    result = math.sqrt(dsquared)
    return result
```

Composition

- Call one function within another function
- Example
  - *Function 1: circle_area()*
    - call Function 2 distance() to calculate radius (center to perimeter node)



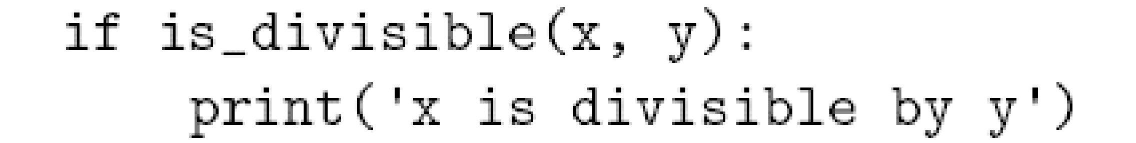
```
def circle_area(xc, yc, xp, yp):
    radius = distance(xc, yc, xp, yp)
    result = area(radius)
    return result
```



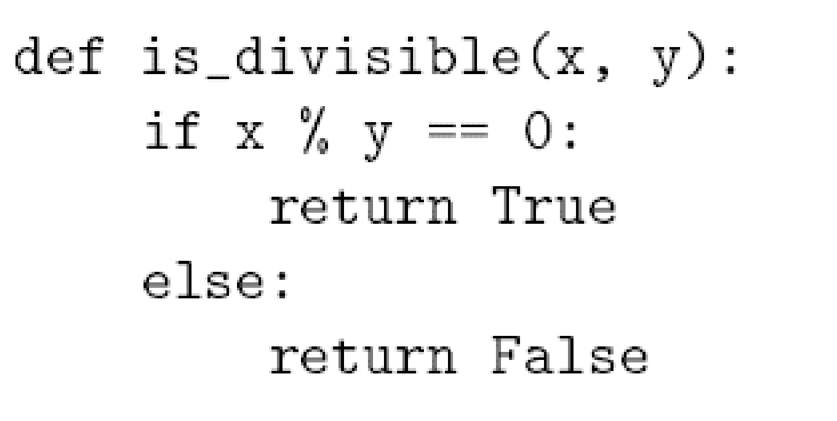
```
def circle_area(xc, yc, xp, yp):
    return area(distance(xc, yc, xp, yp))
```

Boolean Function

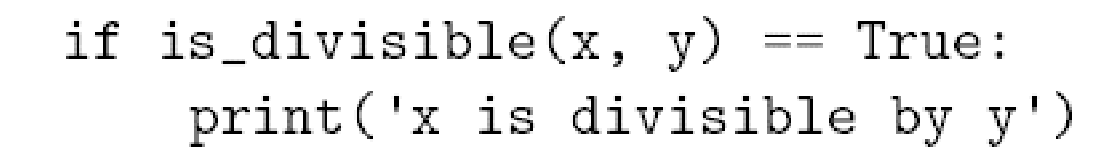
- Boolean
  - *True or False*
- Example
  - *Is x divisible by y?*



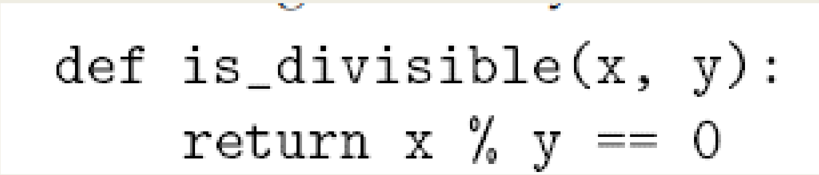
```
if is_divisible(x, y):
    print('x is divisible by y')
```



```
def is_divisible(x, y):
    if x % y == 0:
        return True
    else:
        return False
```



```
if is_divisible(x, y) == True:
    print('x is divisible by y')
```



```
def is_divisible(x, y):
    return x % y == 0
```

Recursion

- Recursive function
  - *A function that call itself (but might have different input parameter) within the function*

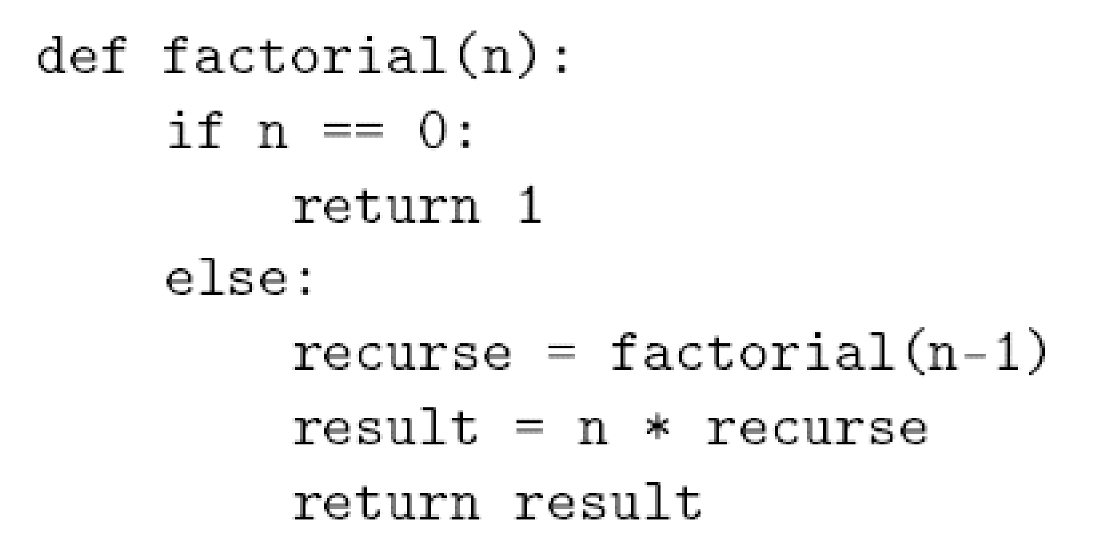
- When to use program with recursion
  - *Solution (e.g. mathematical structure) is in recursive form*
  - *Tips: think/plan before you start typing codes*
    - The goal is problem solving

- Two examples
  - factorial n!
  - Fibonacci series

Recursive Example (I): Factorial

- 0! = 1
- n!=n(n-1)!

F(0)=1
F(n)=F(n-1)*n



```
def factorial(n):
    if n == 0:
        return 1
    else:
        recurse = factorial(n-1)
        result = n * recurse
        return result
```

- Structure
  - *Base case*
  - *General recursive relationship*

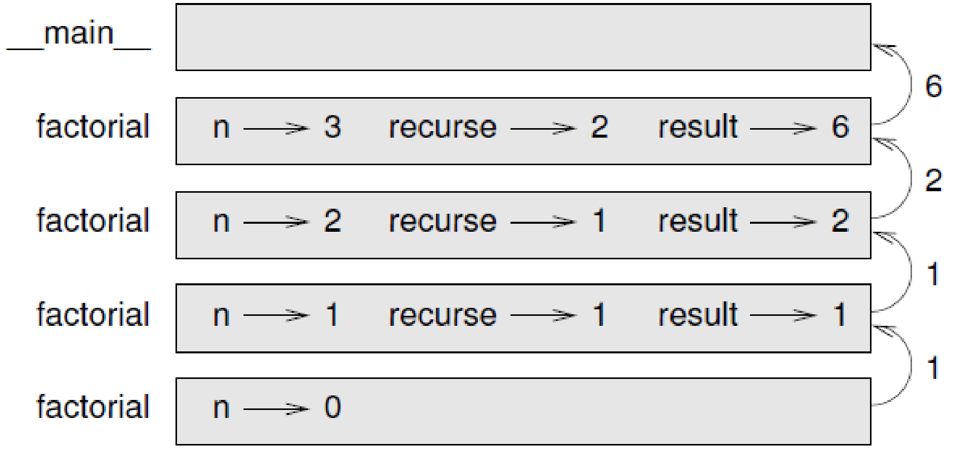
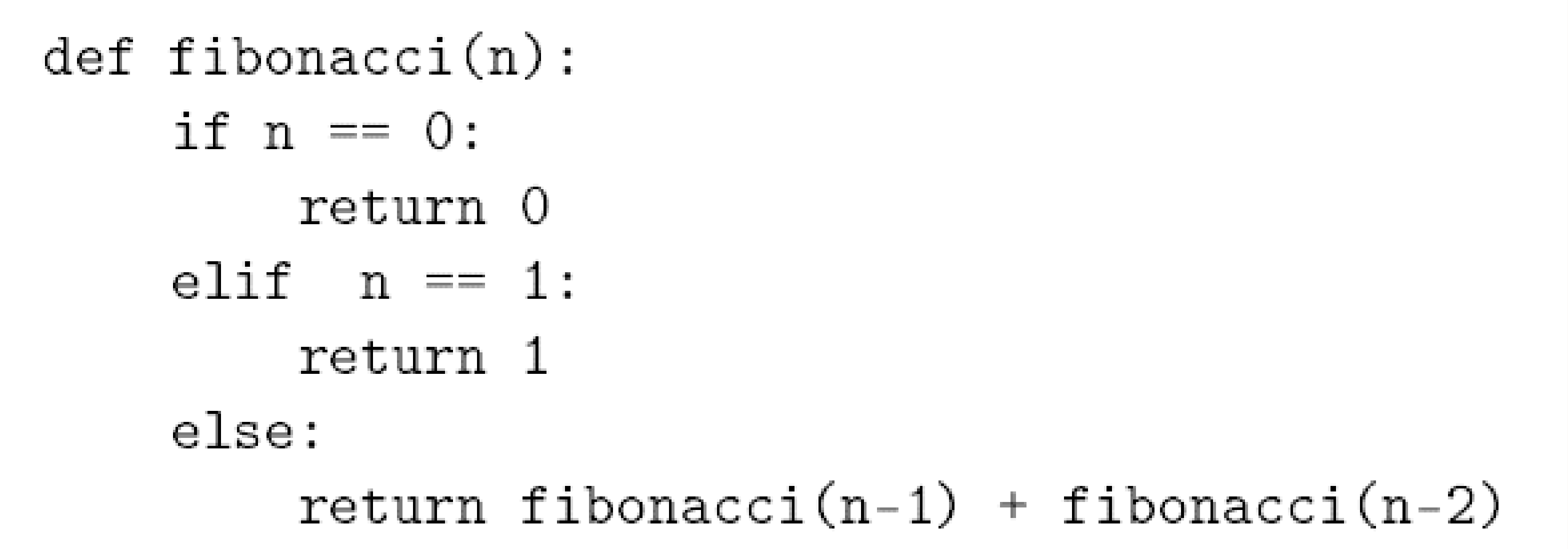


Figure 6.1: Stack diagram.

Recursive Example (II): Fibonacci

- fibonacci(0) = 0
- fibonacci(1) = 1
- fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)

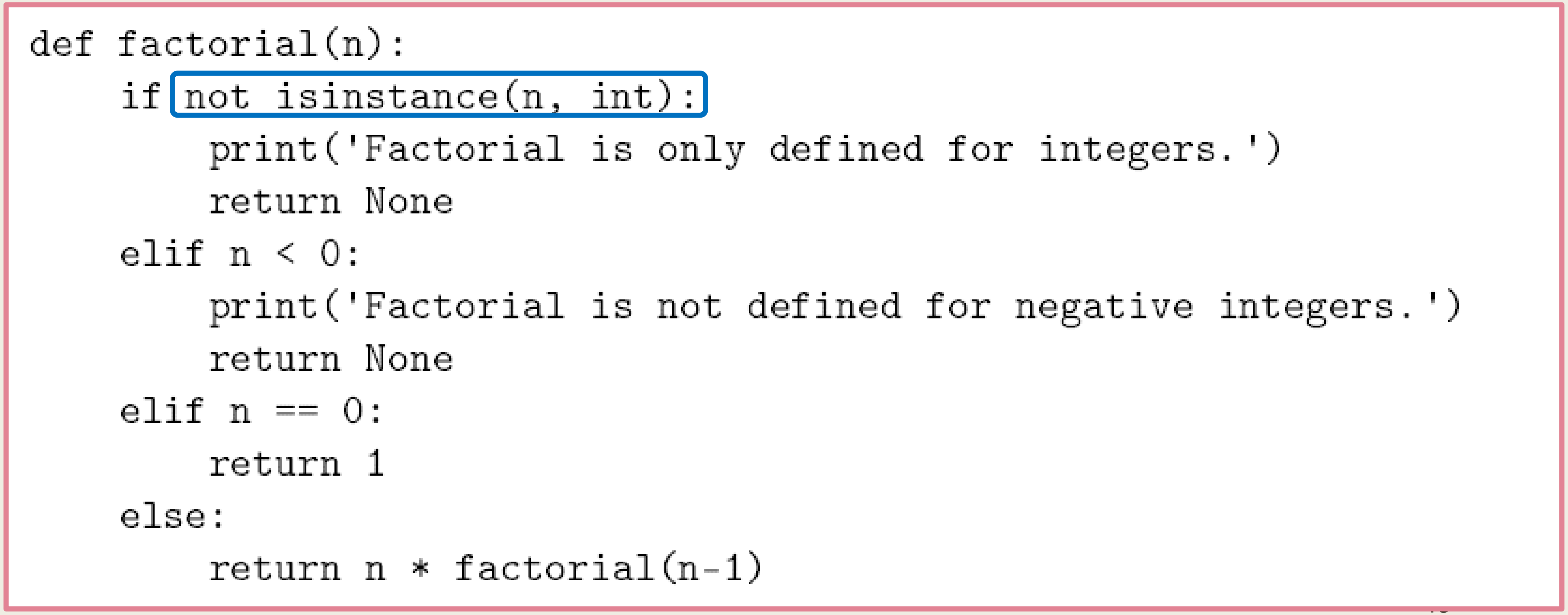
F(0)=0
F(1)=1
F(n)=F(n-1)+F(n-2)



```
def fibonacci(n):
    if n == 0:
        return 0
    elif  n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

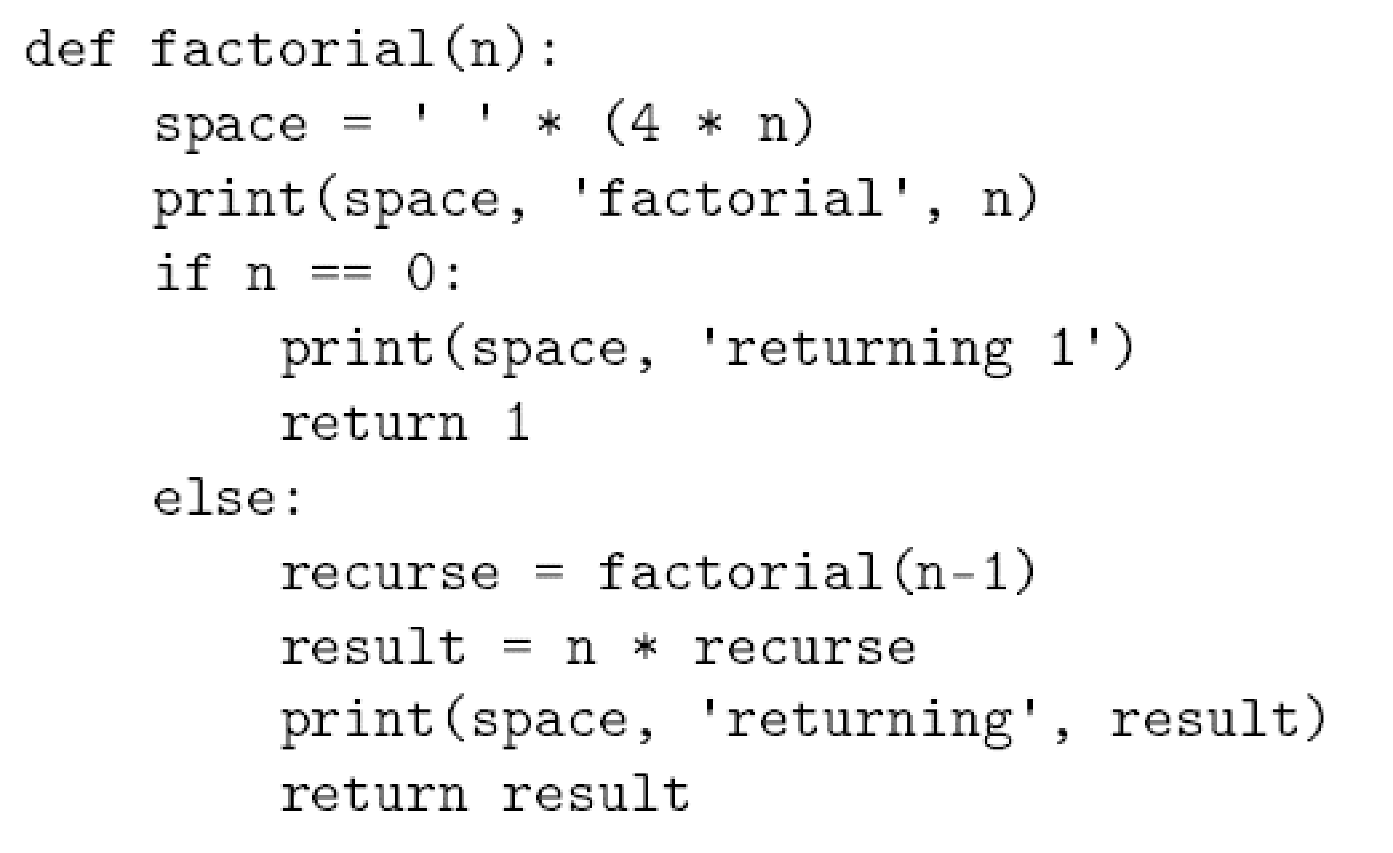
Check Input Validity

- Check variable type
  - *Isinstance (variable, type_to_check)*

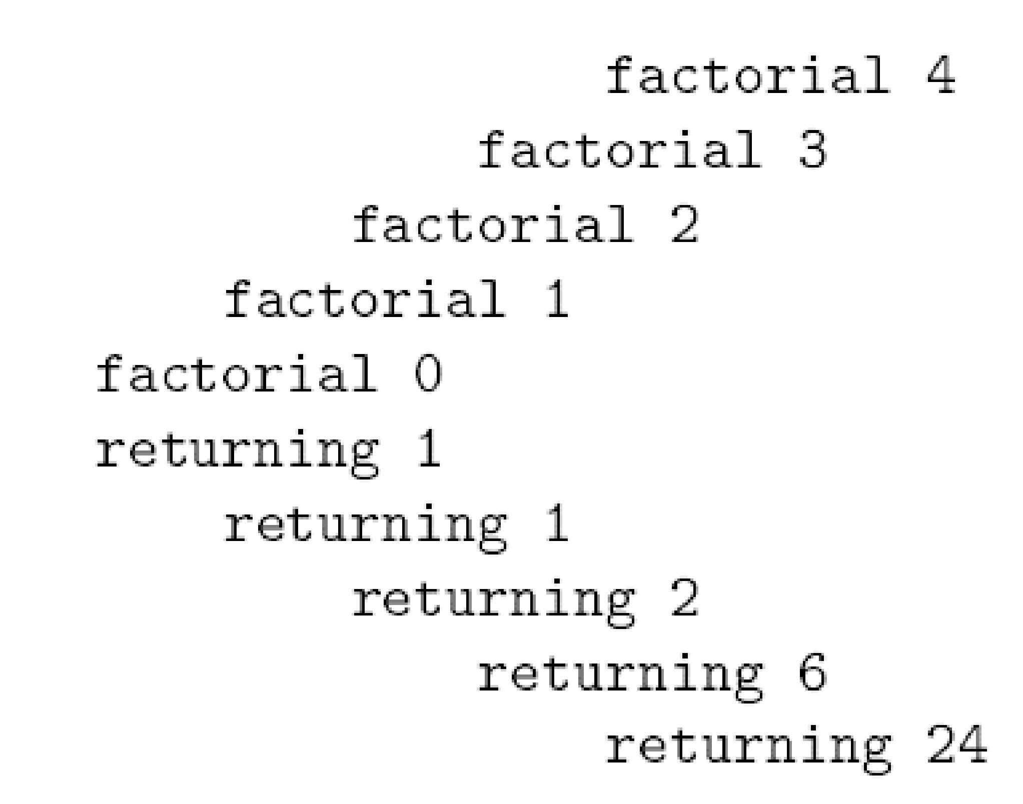


```
def factorial(n):
    if not isinstance(n, int):
        print('Factorial is only defined for integers.')
        return None
    elif n < 0:
        print('Factorial is not defined for negative integers.')
        return None
    elif n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

Example: Extend with fancy printing



```python
def factorial(n):
    space = ' ' * (4 * n)
    print(space, 'factorial', n)
    if n == 0:
        print(space, 'returning 1')
        return 1
    else:
        recurse = factorial(n-1)
        result = n * recurse
        print(space, 'returning', result)
        return result
```

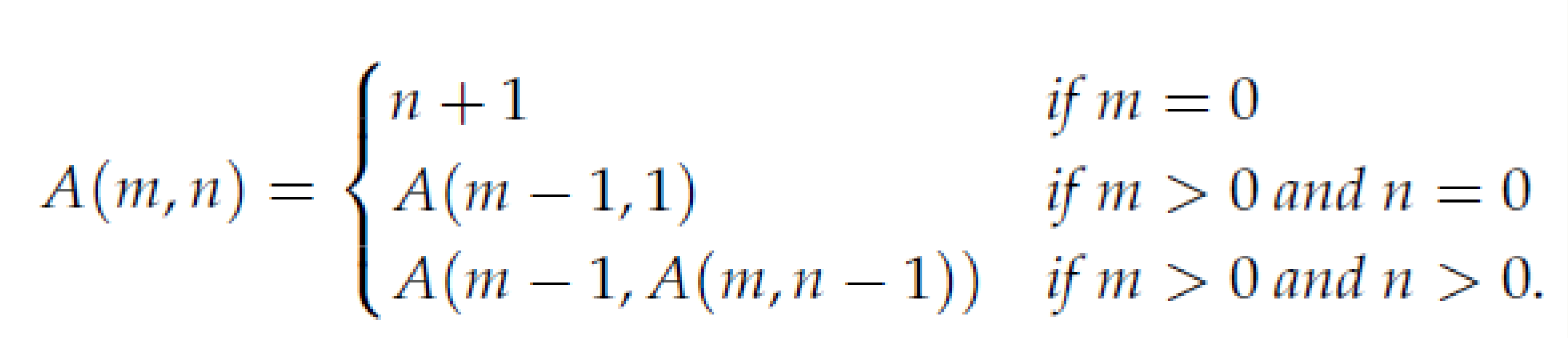


```
                factorial 4
            factorial 3
        factorial 2
    factorial 1
factorial 0
returning 1
    returning 1
        returning 2
            returning 6
                returning 24
```

Reading

- Chapter 6 in textbook “Think Python”

- More recursive example
  - *Ackermann function, A(m,n)*
    - https://en.wikipedia.org/wiki/Ackermann_function



$$A(m,n) = \begin{cases} n+1 & \mathit{if}\ m = 0 \\ A(m-1,1) & \mathit{if}\ m > 0\ \mathit{and}\ n = 0 \\ A(m-1, A(m,n-1)) & \mathit{if}\ m > 0\ \mathit{and}\ n > 0. \end{cases}$$