ISYS90088 Introduction to Application Development

Week10 – Contd. from week 09

Functions

Pseudocode

Dr Thomas Christy

Sem 2, 2017

Pseudocode

Pseudocode is a detailed yet readable description of what a computer program or algorithm must do.

Margaret Rouse - http://www.techtarget.com/contributor/Margaret-Rouse

Pseudocode

- Put water into kettle
- Plug in Kettle
- Put coffee in cup
- Wait for kettle to boil
- Add boiling water to cup
- Add optional milk and sugar
- Stir and serve

Review

Functions: passing arguments to functions

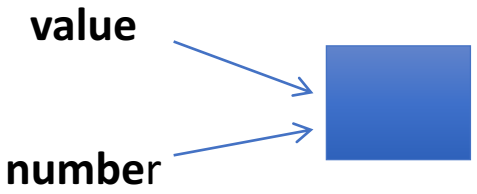
- A argument is any piece of data that is *passed into* a function when the function is called.
- A parameter is a variable that *receives* an argument that is passed into a function.
- Many times we send across pieces of information (data) into a function and tasks are performed within the function.
- And many times information is passed back from a function to the main that called this function using a **`return`** statement .

Example: how to pass values

```
def main():
  value = int(input('enter a number:'))
  show_double(value)

def show_double(number):
   result = number * 2
   print(result)

main()
```



Basic function

```
def name():
        print("Running function")
```

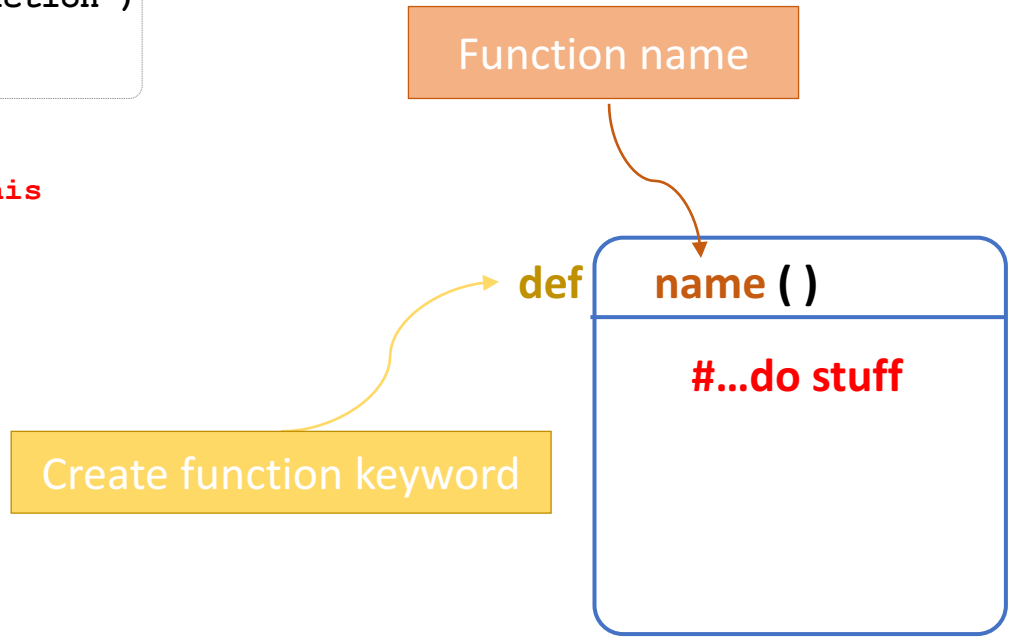
Basic function

```
def name():

        print("Running function")


# to run a function we do this

name()
```



Output:

```
Running function
```

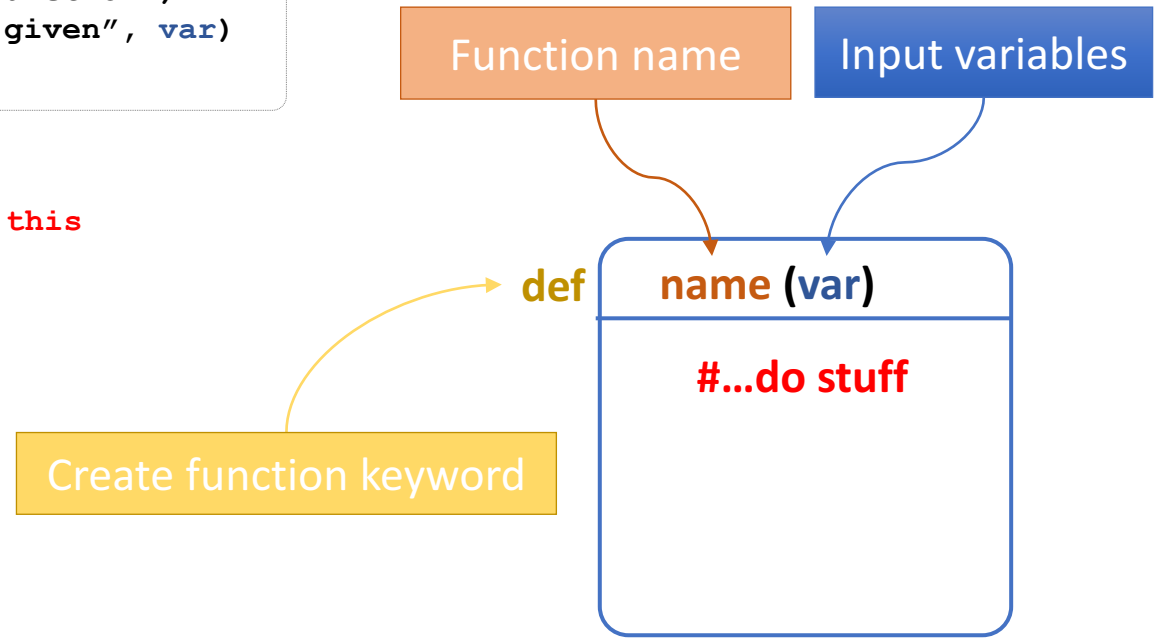
Basic function with parameter

```
def name(var):

        print("Running function")
        print("variable given", var)
```

```
# to run a function we do this

name('Tom')
```



Output:

```
Running function
variable given Tom
```

Basic function with parameter

Parameter

```
def name(var):

        print("Running function")
        print("variable given",
var)
```

Function name

Input variables

```
def name (var)
    #...do stuff
```

Create function keyword

```
# to run a function we do this

name('Tom')
```

Argument

Output:

```
Running function
variable given Tom
```

Basic function with parameter and return value

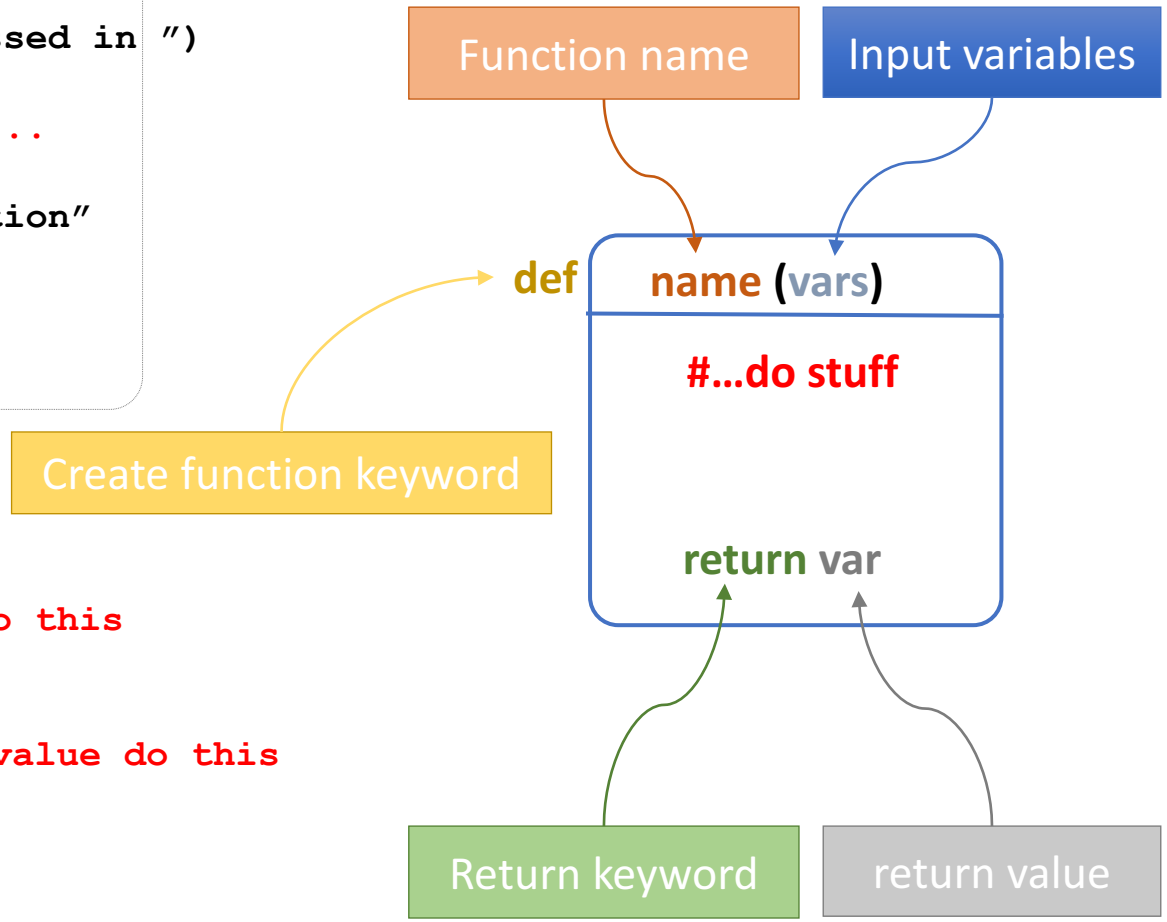
```
def name(vars):

        print(vars, "passed in ")

        #...do anything...

        x = " From Function"
        var = vars + x

        return var
```




```
# to run a function we do this
name('Tom')

# to collect the return value do this
x = name('Tom')

print(x)

# or we can expect the return value like this
print( name('Tom') )
```

Basic function with parameter and return value

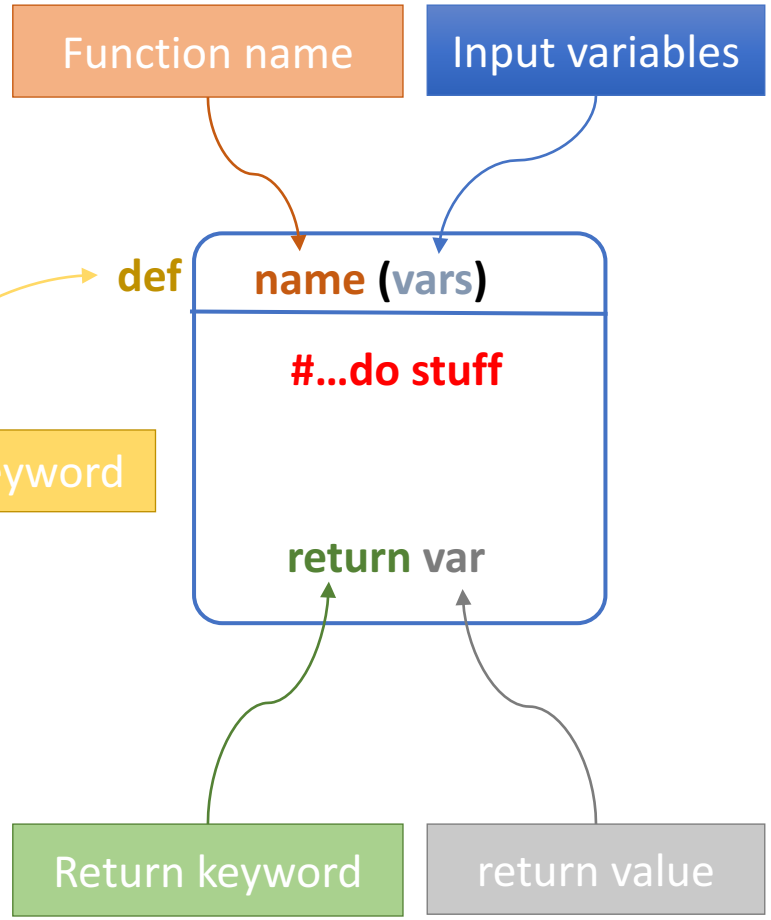
```
def name(vars):

        print(vars, "passed in ")

        #...do anything...

        x = " From Function"
        var = vars + x

        return var
```




```
# to run a function we do this
name('Tom')

# to collect the return value do this
x = name('Tom')

print(x)

# or we can expect the return value like this
print( name('Tom') )
```

Basic function with parameter and return value

```
def name(vars):

        print(vars, "passed
in")

        #...do anything...

        x = " From Function"
        var = vars + x

        return var
```

```
print(name('Tom'))
```

Output:

```
Tom passed in
Tom From Function
```

Input parameters

Basic function

```
def name():

        print("Running function")
```

Basic function with input parameter

```
def name(any_num_of_vars):

        print("Running function")
```

Return

Basic function with return value

```
def name():

        print("Running function")

        return 'something'
```

Basic function with input parameter
and return value

```
def name(any_num_of_vars):

        print("Running function")

        return 'something'
```

Functions and `return` statement

- The value of the expression that follows the key word return will be sent back to part of the program that called this function. This can be any value, expression, or variable that has a value.

- A return statement can also send back:
  - Strings
  - boolean values
  - multiple values
  - dictionaries, etc,

Passing multiple arguments

Example:

Write a program that demonstrates a function that accepts two arguments and then displays their sum.

```
def main():
    print ('the sum of 12 and 45 is')
    show_sum(12, 45)

def show_sum (num1, num2):
    result = num1 + num2
    print(result)

main()
```

Parameter list - The values 12 and 45 are arguments that are passed by *position* to the corresponding parameter variables in the function

Functions and `return` statement

A value-returning function has a `return` statement that returns a value back to the part of the program that called it.

Syntax:

```
def <function_name>():
      statement
      statement
      statement
      return <expression>
```

Functions: simple examples

Print the number of digits in a number

```
def num_digits(n):
        s = str(abs(n))
        print (len(s) - ('.'  in  s))
```

```
def main():
    v = float(input('enter a value:'))
    num_digits(v)

main()
```

Convert from Celsius to Fahrenheit:

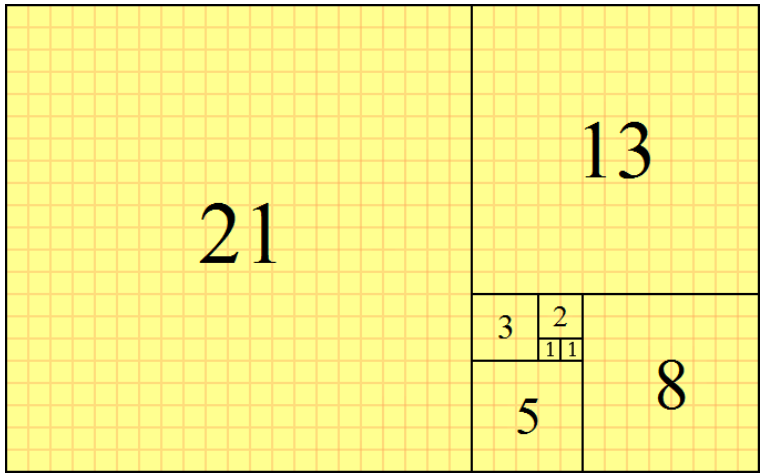
```
def C2F(c):
        print(9*c/5 + 32)
```

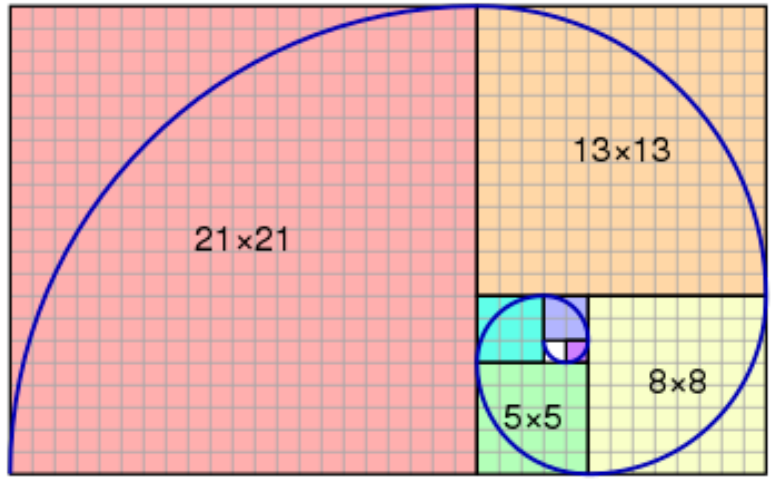
```
def main():
    c = float(input('enter a value:'))
    C2F(c)

main()
```

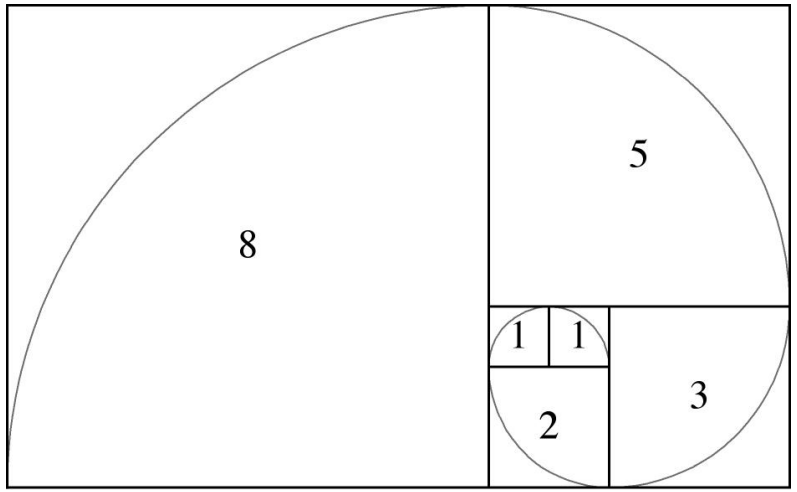
Fibonacci number sequence

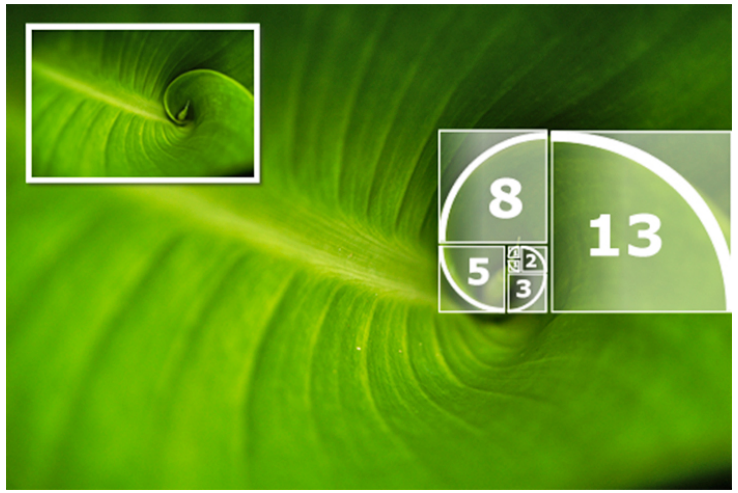
F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765

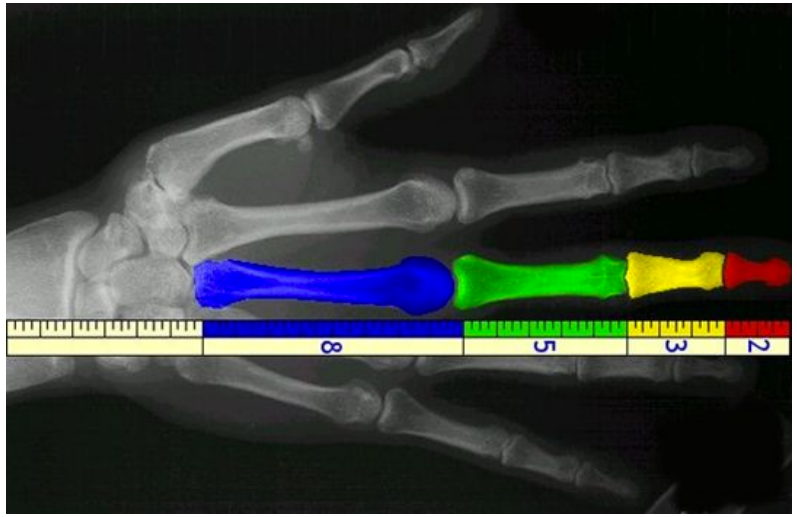


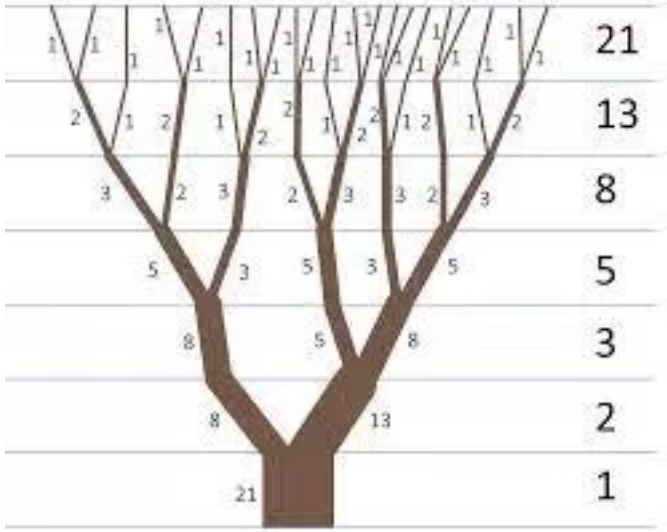


Fibonacci numbers









By Dr. Thomas Christy

Fibonacci

```
# write a function to print the Fibonacci sequence to n values
```

Fibonacci

write a function to print the Fibonacci sequence to n values

```
def f(n):
    a, b = 0, 1
    for i in range(n):
        a, b = b, a + b
        print(a, end=" ")

f(21)
```

Passing multiple arguments

Example:

Write a program that demonstrates a function that accepts two arguments and then displays their sum.

Functions: simple examples

write a function to print the number of digits in a number

write a function to convert from Celsius to Fahrenheit:

Functions: simple examples

write a function to print the number of digits in a number

```
def num_digits(n):
        s = str(abs(n))
        print (len(s) - ('.'  in  s))
```

write a function to convert from Celsius to Fahrenheit:

```
def C2F(c):
        print(9*c/5 + 32)
```

Functions: simple examples

write a function to print the number of digits in a number

```
def num_digits(n):
        s = str(abs(n))
        print (len(s) - ('.'  in  s))
```

write a function to convert from Celsius to Fahrenheit:

```
def C2F(c):
        print(9*c/5 + 32)
```

Now write the main function that calls this function?

Examples 1: return statement

```
# write a program that converts user
input Celsius to Fahrenheit
```

Use Pseudocode:

Examples 1: return statement

```
# write a program that converts user input
Celsius to Fahrenheit


def C2F(c):
      return 9*c/5 + 32



def main():
    cels = int(input('enter a value in Celsius:'))
    f = C2F(cels)
    print(f)

main()
```

Examples 2: return statement

```
# Write a program to calculate the age of two friends inputted
values
```

Use Pseudocode:

Examples 2: return statement

```
def main():
    # get the user's age

    # get the users best friends age

    # get the sum of both ages

    # display the total age



#sum function accepts two int arguments & returns sum of those
arguments



#call main function
```

Examples 2: return statement

```
def main():
    # get the user's age
    first_age = int(input('enter your age:'))
    # get the users best friends age
    second_age = int(input('enter your best friends age:'))
    # get the sum of both ages
    total = sum (first_age, second_age)
    # display the total age
    print('their total age is:', total, 'years old’)


#sum function accepts two int arguments & returns sum of those
arguments
def sum(num1, num2):
    result = num1 + num2
    return result
#call main function
```

Quiz

- What is printed to the screen here?

```
>>>def totallyfy(word):
    return word[:3] + '-totally-' + word[3:]

>>>print(totallyfy('fantastic'))
```

- What is printed to the screen here?

```
def totallyfy(word):
    return word[:3] + '-totally- ' + word[3:]

print(totallyfy(’thomas'))
```

Variables and “Scope”

- Each function (call) defines its own local variable “scope”. Its variables are not accessible from outside the function (call) – so what happens in this example?

```
def subtract_one(k):
      k = k - 1
      return k
i = 0
n = subtract_one(i)
print(i)
print(n)
print(k)
```

Variables and “Scope”

- Are the semantics different to the previous slide?

```
def subtract_one(i):
      i = i - 1
      return i

i = 0
n = subtract_one(i)
print(i)
print(n)
print(k)
```