Functions: Usage

CPSC 217: Introduction to Computer Science for Multidisciplinary Studies I
Fall 2020

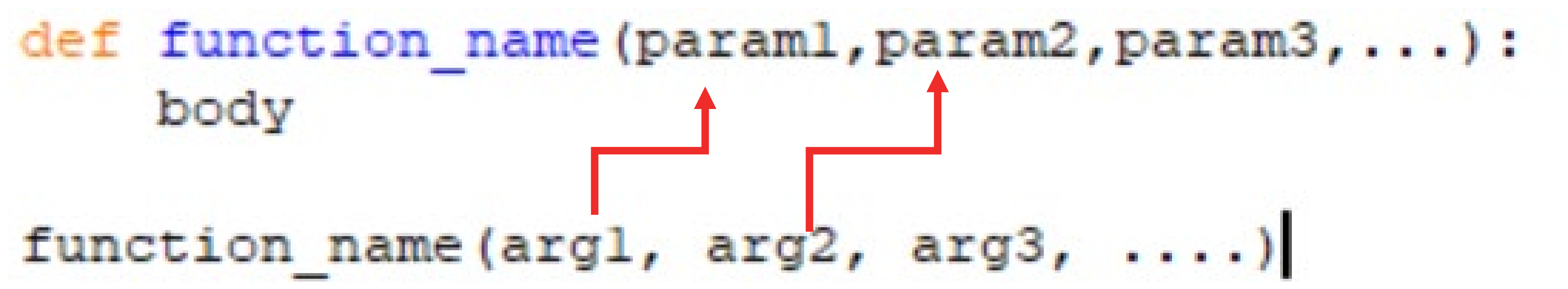
Jonathan Hudson, Ph.D
Instructor
Department of Computer Science
University of Calgary

Tuesday, September 8, 2020

UNIVERSITY OF CALGARY

Function calling review

Select a descriptive name for your function



```
def function_name(param1,param2,param3,...):
    body

function_name(arg1, arg2, arg3, ....)
```

Use brackets when calling functions even if you are not passing any arguments

At least one statement needs to be in a function.

Functions must be defined before they are called!

How to use a function?

- Call function
- Pass valid inputs
- Store the result in a variable

If function returns a value:

```
returnedValue = functionName(values/variables)
```

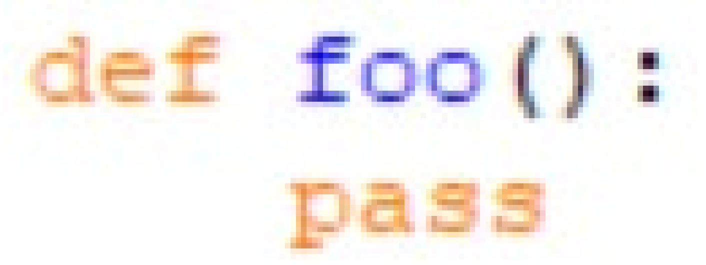
If no value is returned

```
functionName(values/variables)
```

Functions that do nothing

Functions have to have one line of code in them

- Only way to make pythons syntax parsing that is looking for indentation happy
- (Once you put something indented in function the rest of indentation has to match)
  - This is also true for conditionals and loop indentation
- Can use pass keyword to do nothing

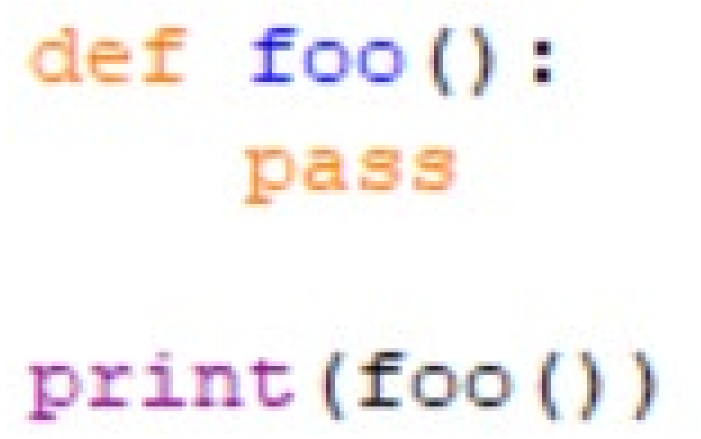


```
def foo():
    pass
```

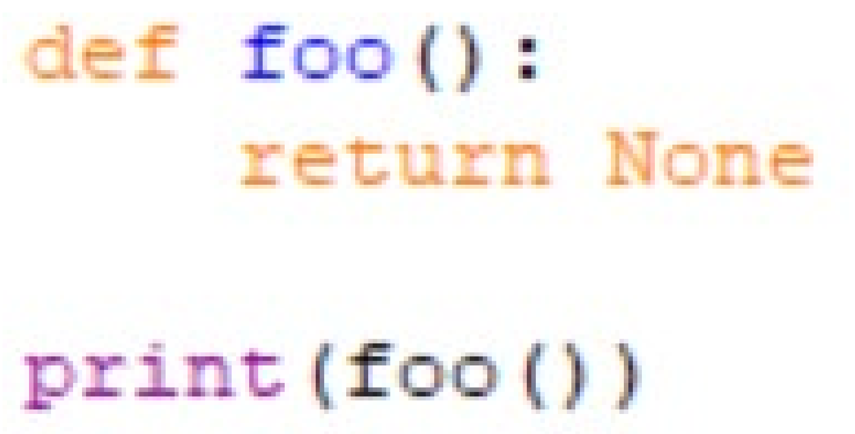
Functions return None by default

Functions in python always return something

- That something is by default nothing or None
- None is a special keyword
- (We often use None in other places in our code to show nothing has been stored in a variable yet)



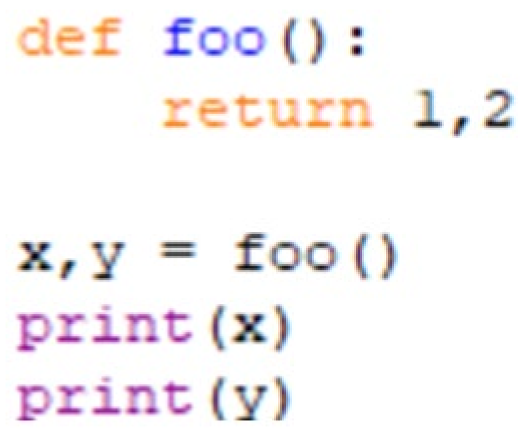
```
def foo():
    pass

print(foo())
```



```
def foo():
    return None

print(foo())
```

Return multiple things

Functions can return multiple things



```
def foo():
    return 1,2

x,y = foo()
print(x)
print(y)
```

Return values

- Format

```
def <function name> (param1, param2, ...):
        body
        return var1, var2, ...
```

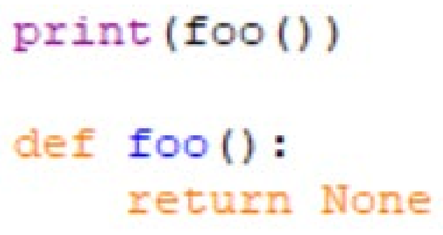
- The results can be stored into variables for later use

```
var1, var2, ... = <function name> (arg1, arg2, ...)
```

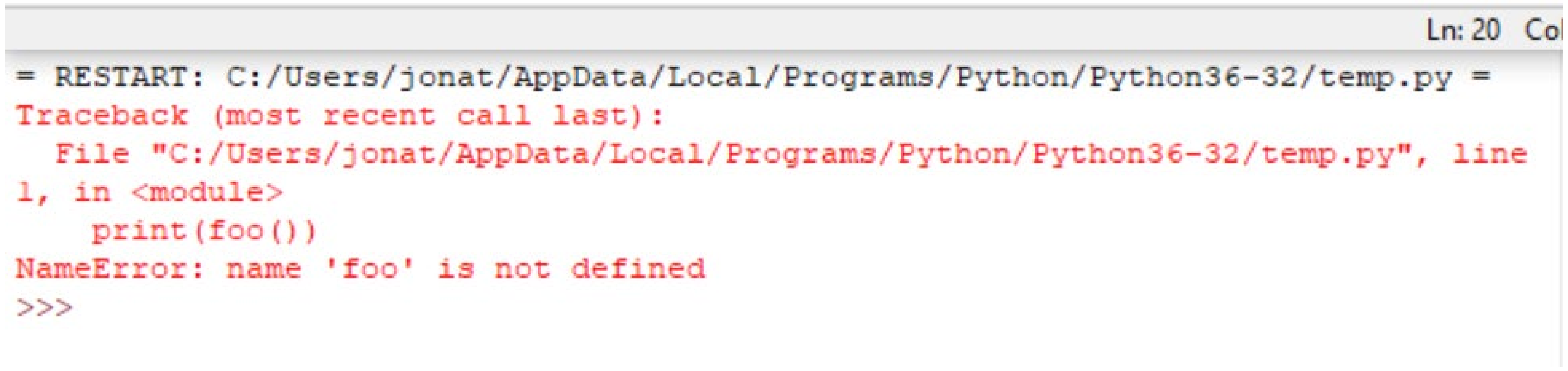
Namespace

Must define functions before use

Functions must be declared before use



```python
print(foo())

def foo():
    return None
```



```
= RESTART: C:/Users/jonat/AppData/Local/Programs/Python/Python36-32/temp.py =
Traceback (most recent call last):
  File "C:/Users/jonat/AppData/Local/Programs/Python/Python36-32/temp.py", line
1, in <module>
    print(foo())
NameError: name 'foo' is not defined
>>>
```

Examples

UNIVERSITY OF CALGARY

Some simple functions

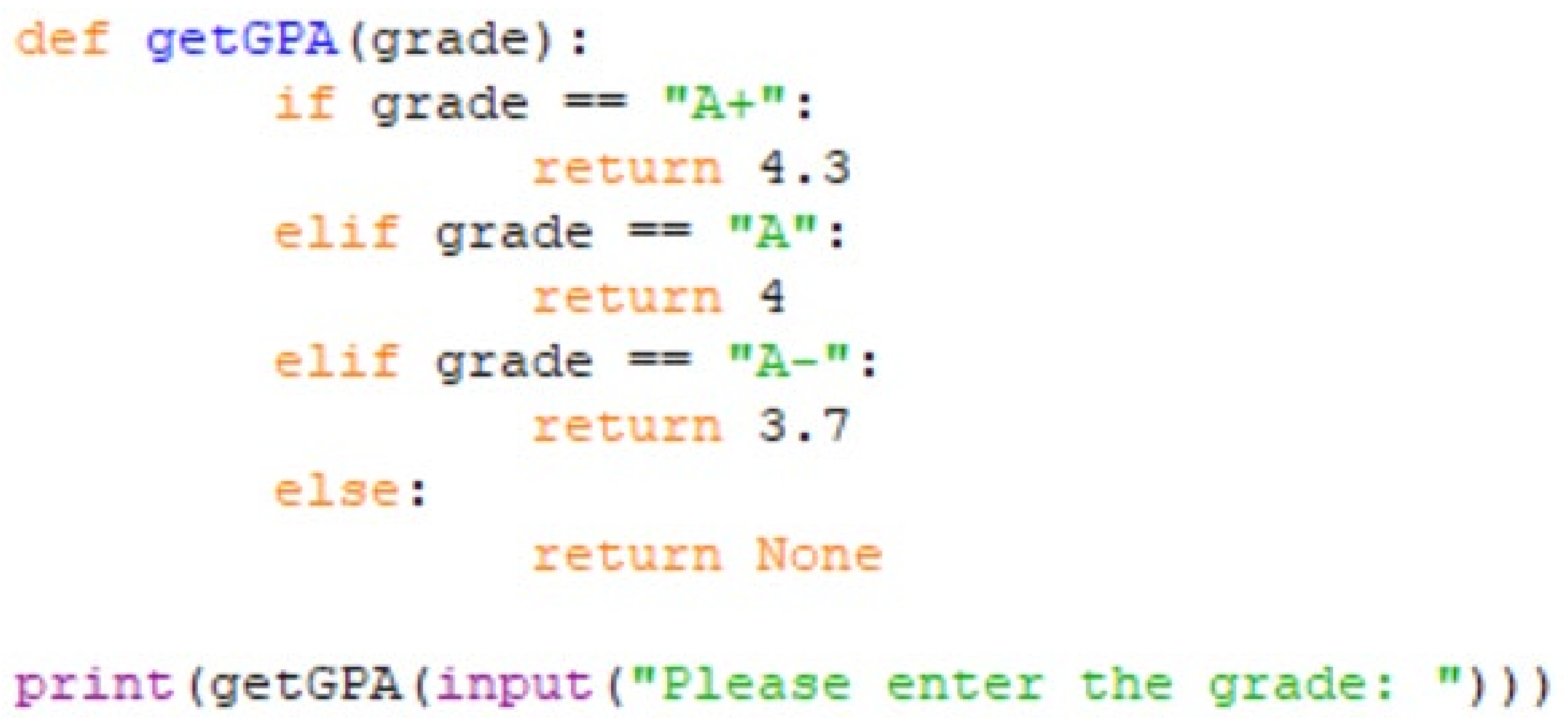
```python
import math

def CircleArea(radius):
    return(math.pi* radius**2)

print(CircleArea(10))
```

```python
def sumTo(n):
    return((n * (n + 1)) / 2)

print(sumTo(10))
```

```python
def IsEven(iNumber):
    return (iNumber % 2 == 0)


def IsOdd(iNumber):
    return (iNumber % 2 != 0)

print(IsEven(50))
print(IsOdd(50))
```

Design

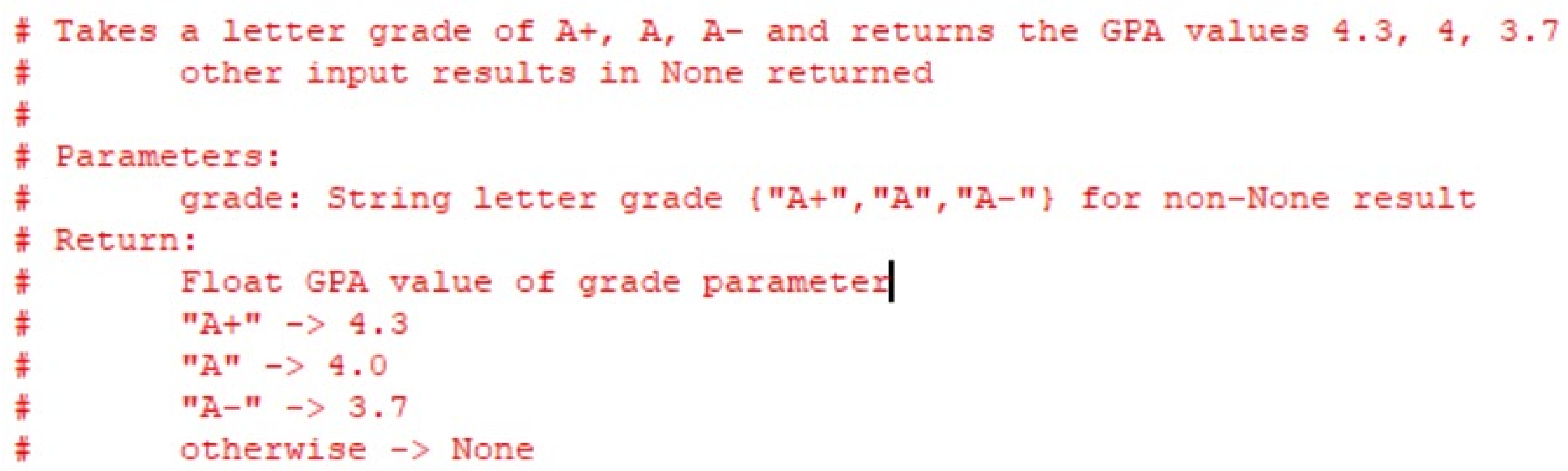
There are challenges in defining a function



```
def getGPA(grade):
        if grade == "A+":
                return 4.3
        elif grade == "A":
                return 4
        elif grade == "A-":
                return 3.7
        else:
                return None

print(getGPA(input("Please enter the grade: ")))
```

User-Defined Functions - Commenting

- A good function always contains explicit comments that describe the purpose of the function, the parameters, and returned values.



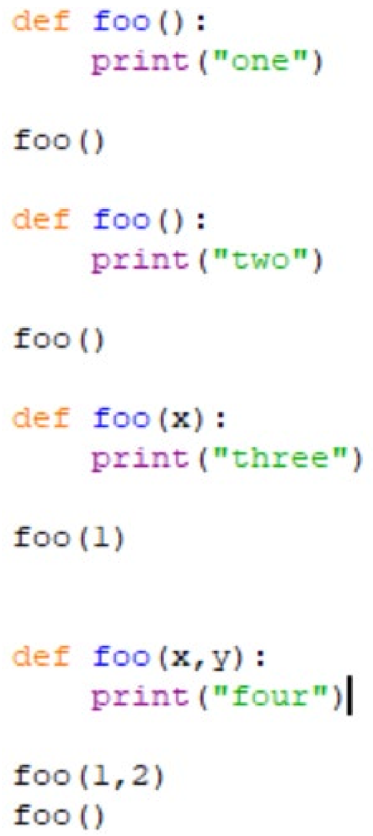
```
# Takes a letter grade of A+, A, A- and returns the GPA values 4.3, 4, 3.7
#       other input results in None returned
#
# Parameters:
#       grade: String letter grade {"A+","A","A-"} for non-None result
# Return:
#       Float GPA value of grade parameter
#       "A+" -> 4.3
#       "A" -> 4.0
#       "A-" -> 3.7
#       otherwise -> None
```

Namespace

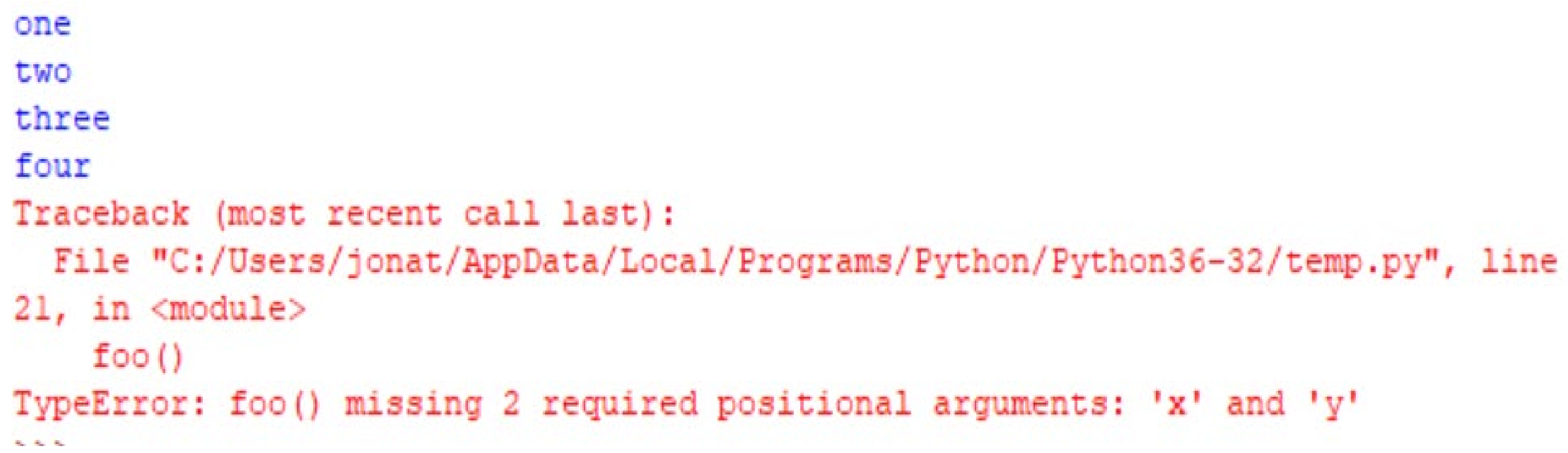
Re-defining functions

Dangers of functions (re-use name)

- Python only lets you have one function per name, but you can override previous usage (ignores parameters unlike other languages)



```python
def foo():
    print("one")

foo()

def foo():
    print("two")

foo()

def foo(x):
    print("three")

foo(1)


def foo(x,y):
    print("four")

foo(1,2)
foo()
```



```
one
two
three
four
Traceback (most recent call last):
  File "C:/Users/jonat/AppData/Local/Programs/Python/Python36-32/temp.py", line
21, in <module>
    foo()
TypeError: foo() missing 2 required positional arguments: 'x' and 'y'
```

Parameter order

Calling Functions - Order of Parameters

- Function parameters are position sensitive.
- When calling a function that accepts parameters, make sure your arguments are in the same order of the parameters.
- **WARNING:** Not following the order of the parameters will result in parameters having wrong values, which may lead to semantic and runtime errors.

```
def printbar(char, num = 10):
     bar = ''
     for i in range(num + 1):
          bar = bar + char
     print(bar)

printbar(20, '=')
```

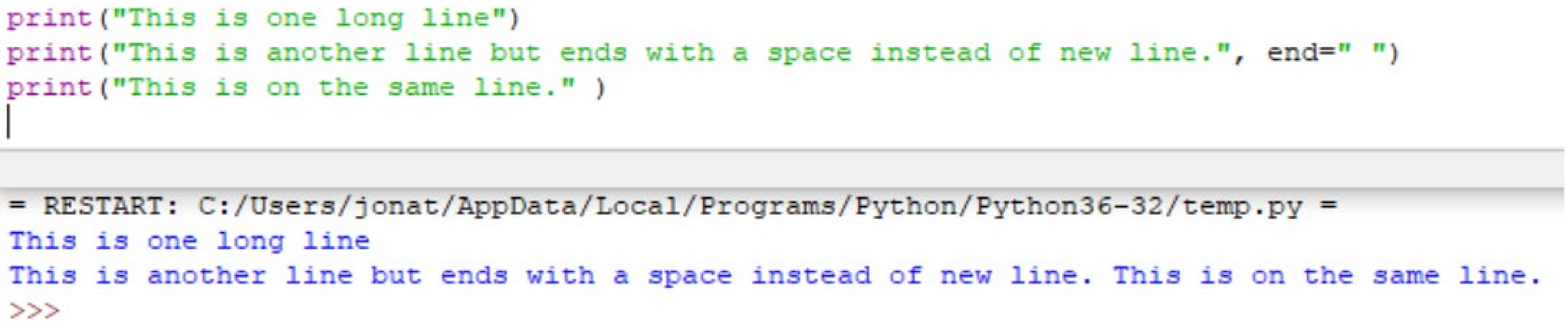
Keyword parameters

- Keyword parameters allow us to match arguments with parameters by name, instead of positions

```
def payroll (rate, hours):
        bounus = 5
        salary = rate * (hours + bounus)
        return (salary)


payment = payroll (hours = 40, rate = 15)
print ("$%d has been paid." % (payment))
```

```
$675 has been paid.
```

We can do this with functions you already use



```
print("This is one long line")
print("This is another line but ends with a space instead of new line.", end=" ")
print("This is on the same line." )
```

```
= RESTART: C:/Users/jonat/AppData/Local/Programs/Python/Python36-32/temp.py =
This is one long line
This is another line but ends with a space instead of new line. This is on the same line.
>>>
```

Scope

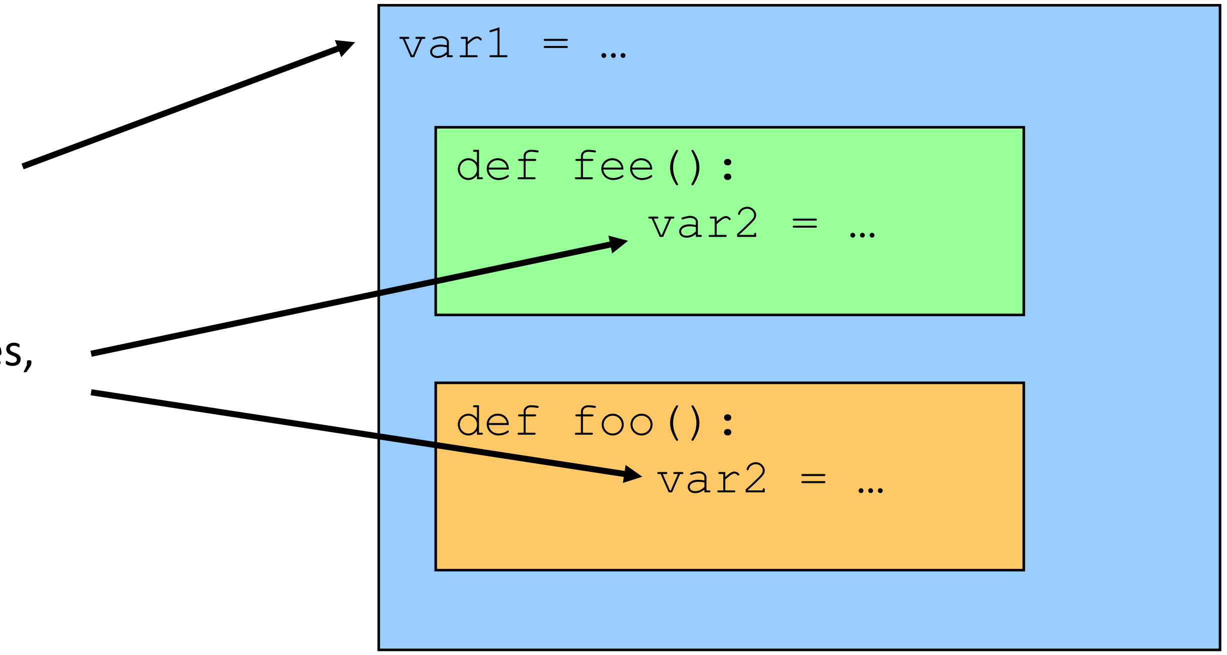
UNIVERSITY OF CALGARY

Scope of Variables

- Variables are memory locations that are used for the temporary storage of data
- The scope of a variable is the section of code in which it is accessible

The global scope:
Accessible by both functions

Local scopes:
Two different memory spaces,
Accessible only within their
functions



Scope of Variables - Local Variables

- Local variables are only accessible to the function where they are defined.
- The memory for local variables is only allocated (reserve the memory) when the function is running and deallocated (free up the memory) when the function reaches the end.

- Local variables are defined (memory allocated and value stored) each time the function is called.

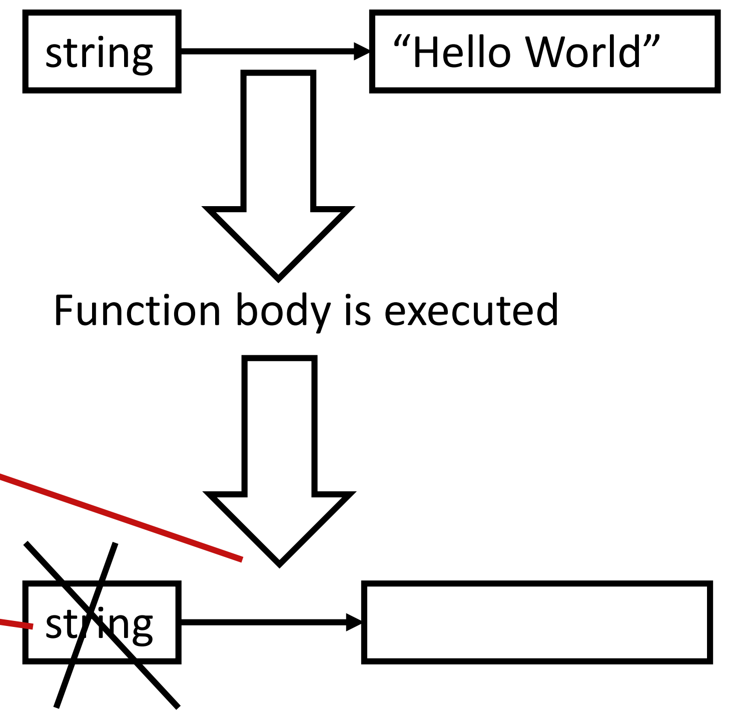
Scope of Variables - Local Variables

```
def foo():
    string = "Hello World!"
    print(string)
```

string is a local variable

Function is done

Memory deallocated



Scope of Variables - Global Variables

- Variables that are declared within the body of a function have a **local scope** → Accessible from inside the function only
    - This includes the parameters

- Variables that are declared outside the body of a function have a **global scope** → Accessible from anywhere in the program

- In Python, global variables can only be modified in global scope.
- They cannot be modified in local scope unless the global keyword is used:
    - **global variableName**

Scope of Variables - Global Variables

```
def failedChange():
        someGlobalVar = "Without Using Global Keyword"

def successfulChange():
        global someGlobalVar
        someGlobalVar = "Using Global Keyword"


someGlobalVar = "I am Global"
print(someGlobalVar)
```

```
I am Global
```

```
failedChange()
print(someGlobalVar)
```

```
I am Global
```

```
successfulChange()
print(someGlobalVar)
```

```
Using Global Keyword
```

Scope of Variables - Variable lifetime

- The lifetime of a variable is the time that a variable is allocated a memory space.
- The memory is allocated at the time of variable declaration

- **Global variables** exist until the program terminates
- **Local variables** exist until the function containing it finishes

Program Structure – Functions

UNIVERSITY OF CALGARY

Structure

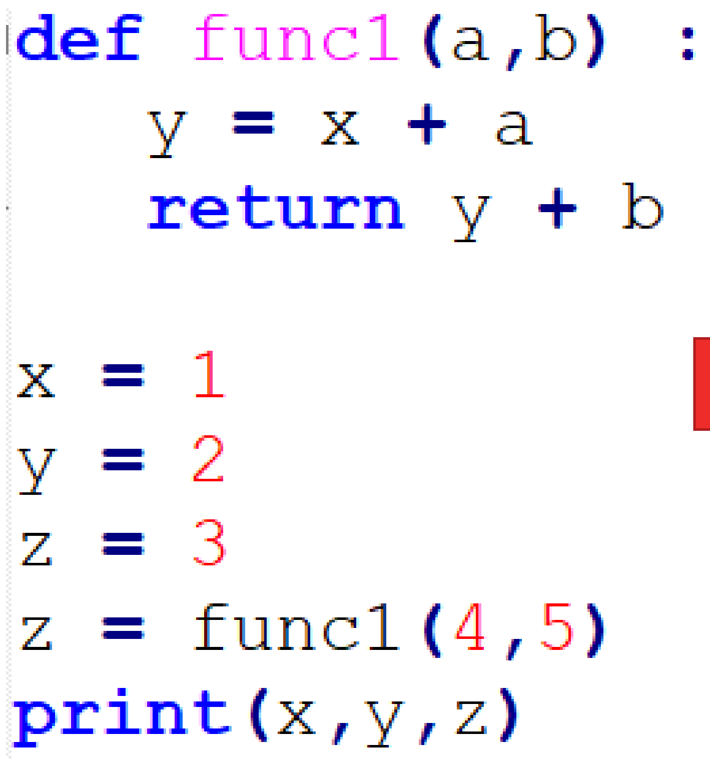
```
def func1():

def func2():

def func3():

…
def main():
            func1()
            func2()
            …

main()
```

The main function

The only code outside functions

Function Tracing

Scope



```
def func1(a,b) :
    y = x + a
    return y + b

x = 1
y = 2
z = 3
z = func1(4,5)
print(x,y,z)
```

```
1 2 10
```

Scope

```
def func1(x,y) :
    return x + y

def func2(x,y) :
    return x * y

def func3(x,y) :
    return func1(x,y) - func2(1,y)

def main() :
    print(func3(1,2))

main()
```

```
1
```

Trace the code

```
def numbers(a,b):
    counter = 1
    while(a != b):
        print(counter)
        #counter += 1
        counter = counter + 1
        if a > b:
            a = a - b
        else:
            b = b - a
    return a
print(numbers(12,15))
```

```
1
2
3
4
3
```

Onward to … lists, dictionaries, and strings.

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~hudsonj/

UNIVERSITY OF CALGARY