ShiP.py

Learn to Py while Shelter-in-Place

L5: Functions

A volunteering educational initiative during COVID-19

ShiP Crew

JD

Teddy

Chinmay

Pratik

Siddharth

Umang

Waseem

A volunteering educational initiative during COVID-19

Topics

PHASE I: Foundations

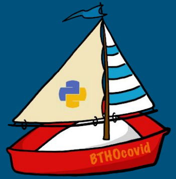
All times are in CDT (GMT-5)

1.	Variables, Expressions, Simple I/O	Sat, April 18 (11 am-12 noon)	☀
2.	Boolean Decisions (branching)	Wed, April 22 (9 pm-10 pm)	🌙
3.	Repetitions (loops)	Sat, April 25 (11 am-12 noon)	☀
4A.	Collective Data Structures (Lists and Tuples)	Wed, April 29 (9 pm-10 pm)	🌙
4B.	CDS (Dictionaries and Sets)	Sat, May 02 (11 am-12 noon)	☀
5.	Functions	Wed, May 06 (9 pm-10 pm)	🌙
6.	File I/O	Sat, May 09 (11 am-12 noon)	☀

Lecture 5

AGENDA

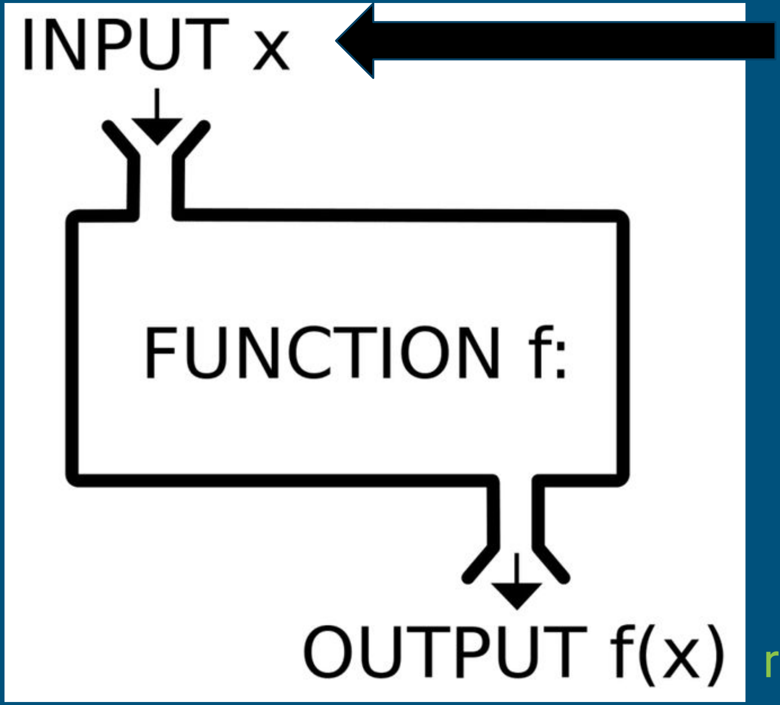
- Defining a function
- Function call
- Recursive function
- Parameters vs arguments
- Types of parameters



How does a Toaster function ?

What is a function ?

A reusable code that takes an input, performs computation and gives you an output



Parameters

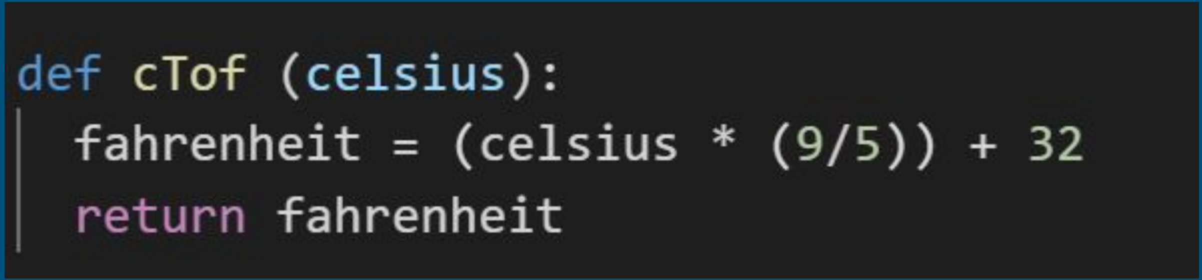
return

When are functions useful?

- Let's say you need to convert celsius to fahrenheit multiple times in your program from different locations in the code

- Instead of writing the formula again and again, you can reduce the redundancy by defining a function and use that function whenever required

- This makes the code modular, enhances code maintenance and helps us follow the DRY (Don't Repeat Yourself) principle in programming

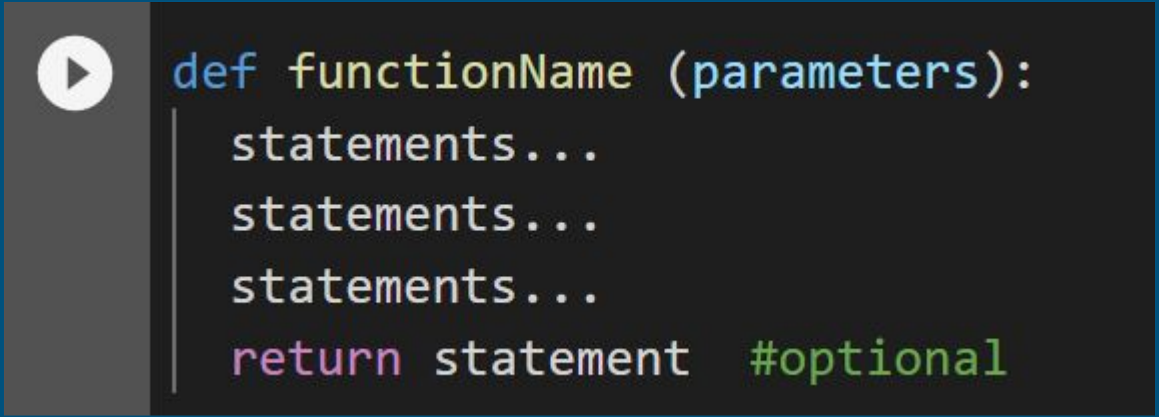


```
def cTof (celsius):
  fahrenheit = (celsius * (9/5)) + 32
  return fahrenheit
```

Defining a function in python

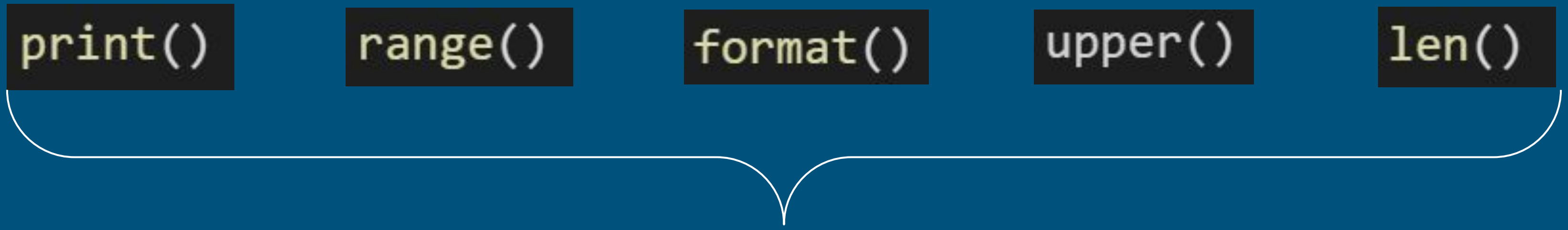
A function is defined in python using the def keyword

Syntax



```
def functionName (parameters):
  statements...
  statements...
  statements...
  return statement  #optional
```

Examples of functions - inbuilt



```
print()
```

```
range()
```

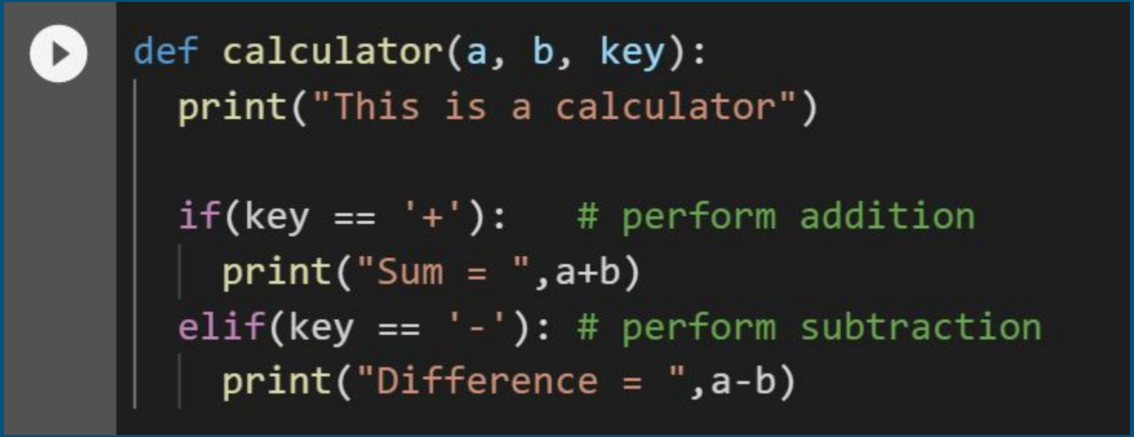
```
format()
```

```
upper()
```

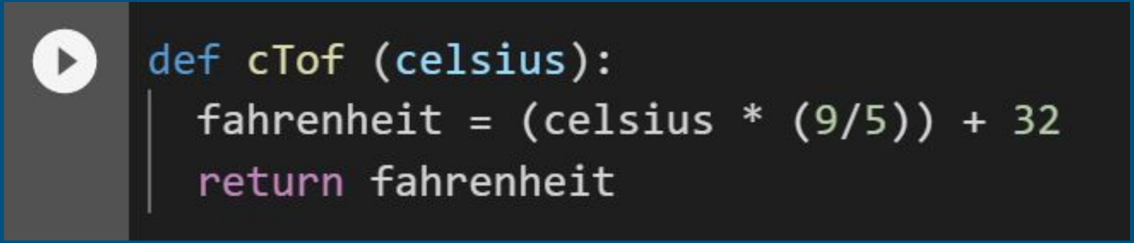
```
len()
```

Some of Python’s inbuilt functions

Examples of user defined functions



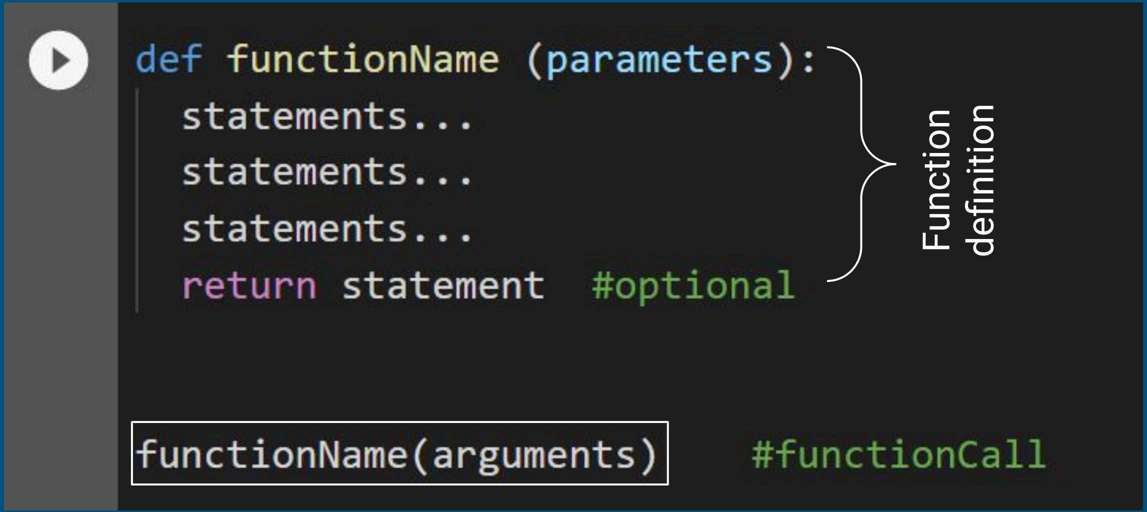
```python
def calculator(a, b, key):
  print("This is a calculator")

  if(key == '+'):   # perform addition
    print("Sum = ",a+b)
  elif(key == '-'): # perform subtraction
    print("Difference = ",a-b)
```



```python
def cTof (celsius):
  fahrenheit = (celsius * (9/5)) + 32
  return fahrenheit
```

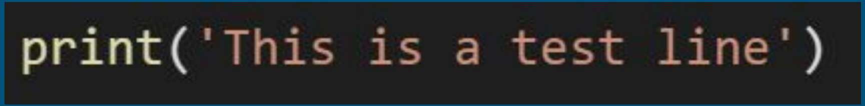
Calling a function

- Just defining a function doesn't actually execute the function
- It must be called from within your program after it has been defined
- A function is called using its name followed by ( )

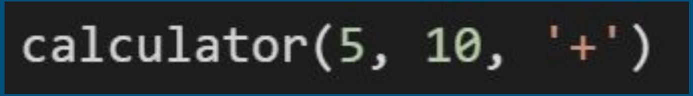


```
def functionName (parameters):
    statements...
    statements...
    statements...
    return statement  #optional


functionName(arguments)     #functionCall
```

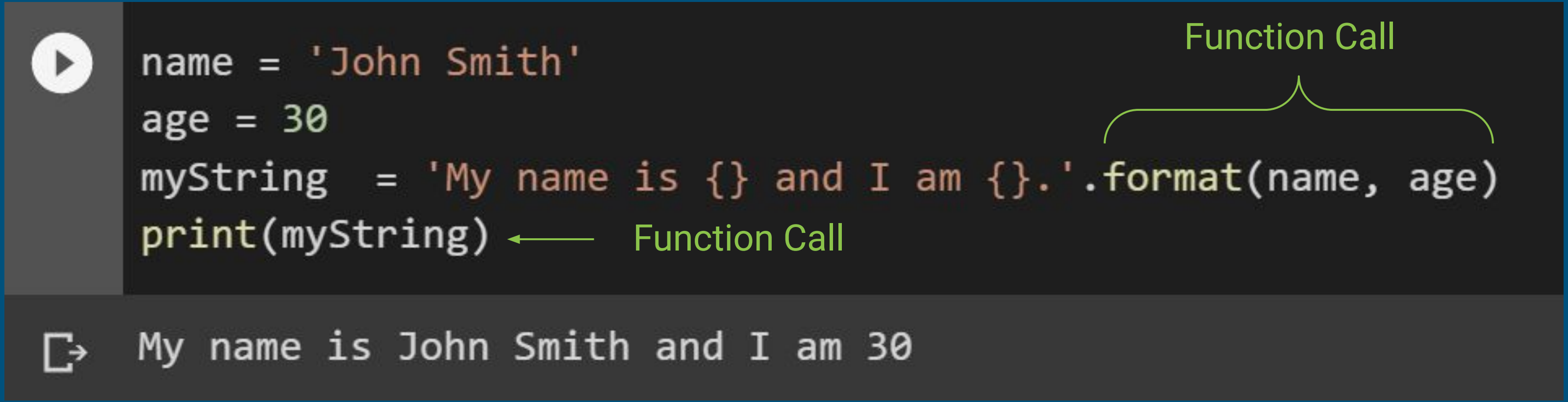
Examples of function call



```
print('This is a test line')
```



```
calculator(5, 10, '+')
```

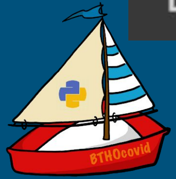


```
name = 'John Smith'
age = 30
myString  = 'My name is {} and I am {}.'.format(name, age)
print(myString)
```

Function Call

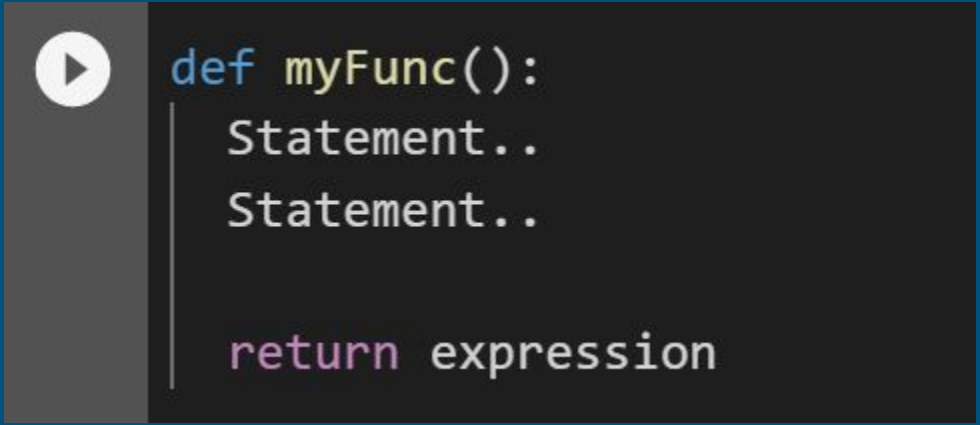
Function Call

```
My name is John Smith and I am 30
```



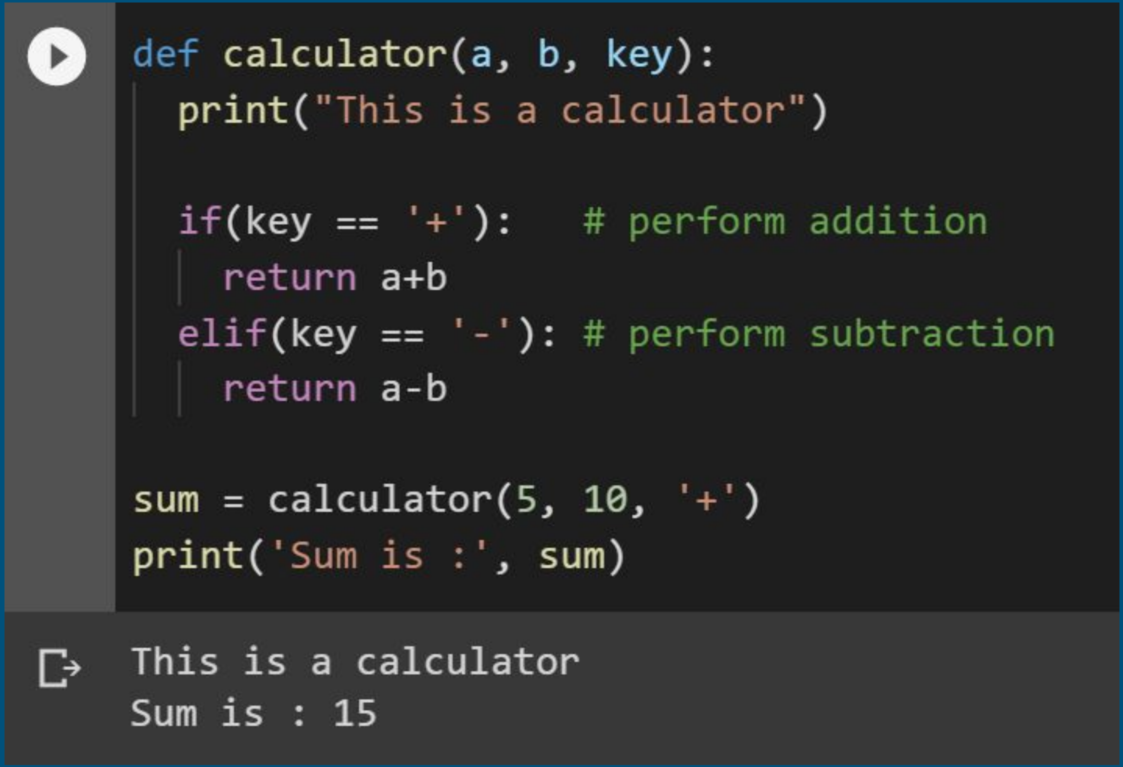
Return statement

- A **return** statement is used to return a value to the caller of the function (optional)
- Any function statements after return are not executed



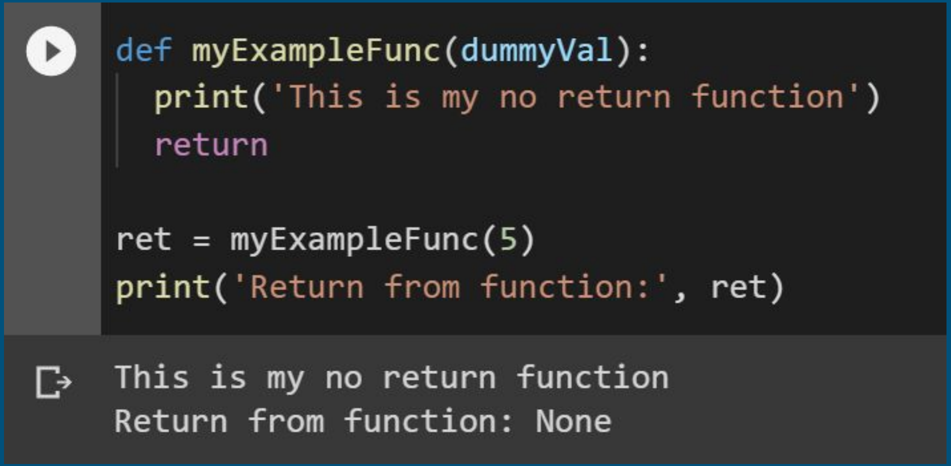
```
def myFunc():
  Statement..
  Statement..

  return expression
```

Example



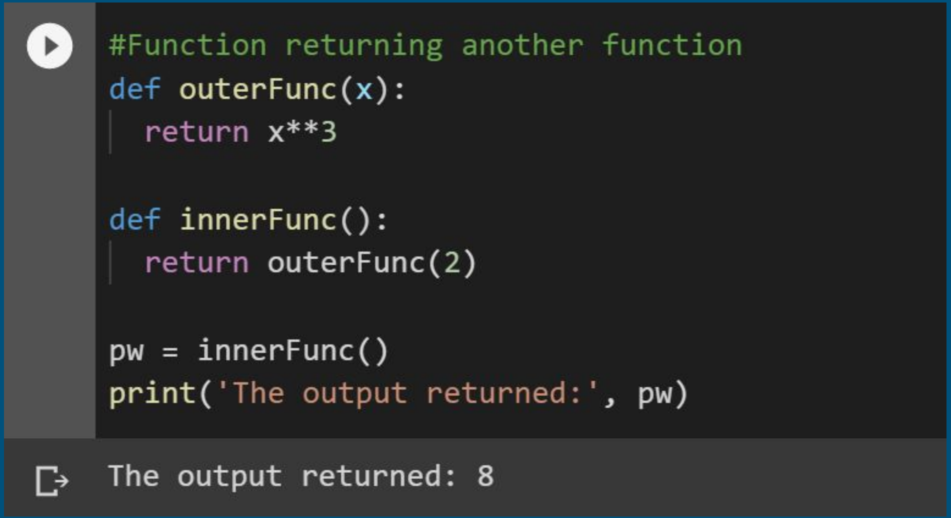
```
def calculator(a, b, key):
  print("This is a calculator")

  if(key == '+'):   # perform addition
    return a+b
  elif(key == '-'): # perform subtraction
    return a-b

sum = calculator(5, 10, '+')
print('Sum is :', sum)
```

```
This is a calculator
Sum is : 15
```



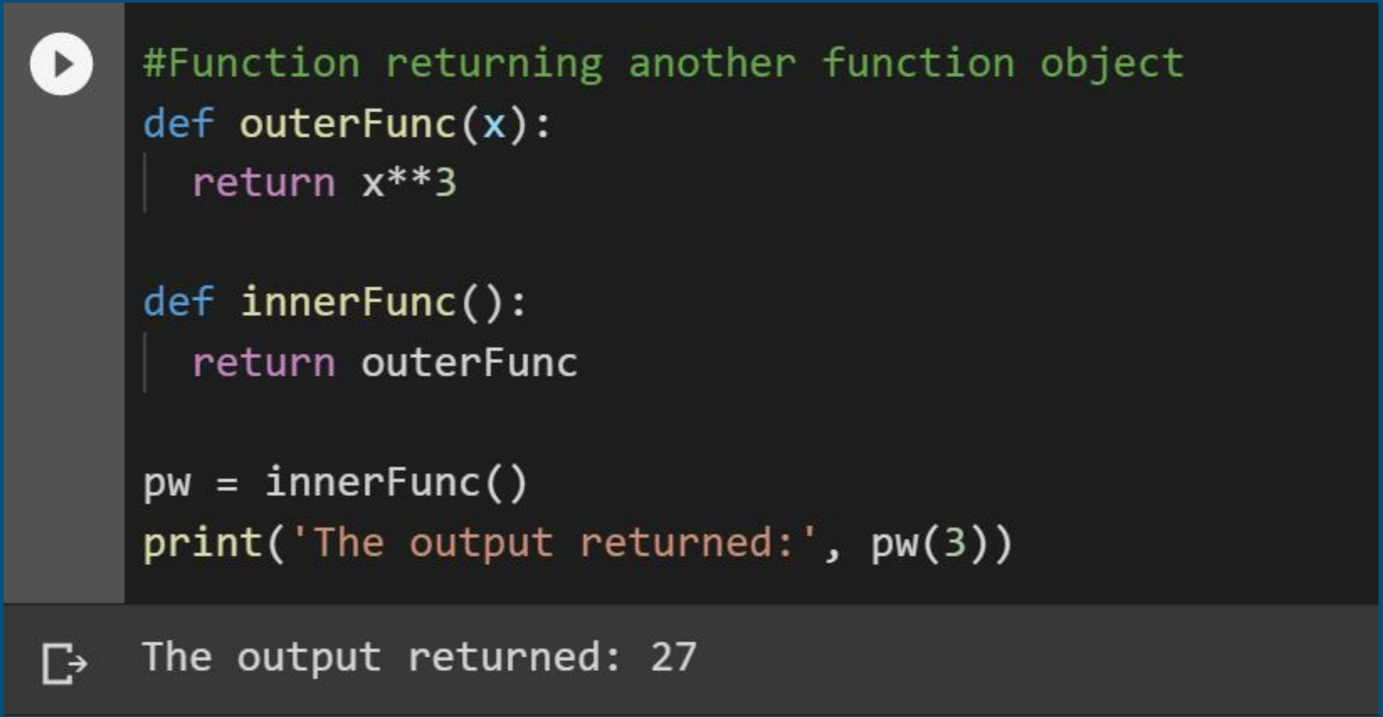
```python
def myExampleFunc(dummyVal):
  print('This is my no return function')
  return

ret = myExampleFunc(5)
print('Return from function:', ret)
```

```
This is my no return function
Return from function: None
```



```python
#Function returning another function
def outerFunc(x):
  return x**3

def innerFunc():
  return outerFunc(2)

pw = innerFunc()
print('The output returned:', pw)
```

```
The output returned: 8
```



```python
#Function returning another function object
def outerFunc(x):
  return x**3

def innerFunc():
  return outerFunc

pw = innerFunc()
print('The output returned:', pw(3))
```

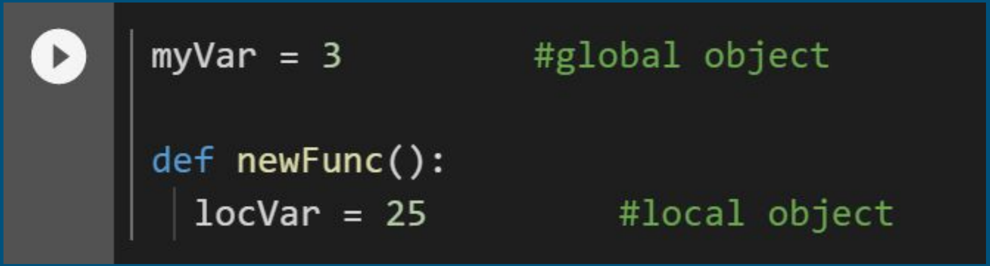
```
The output returned: 27
```

Scope of Objects: Local vs Global

Refers to the places in the code where you can see or access the object. There are two types of scopes:

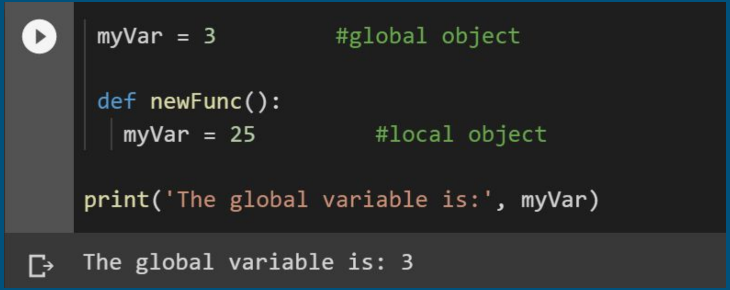
- Global: If you define a variable (object) in the main body scope anywhere outside a function definition

- Local: A variable (object) defined inside the function scope. It can only be accessed from within the function



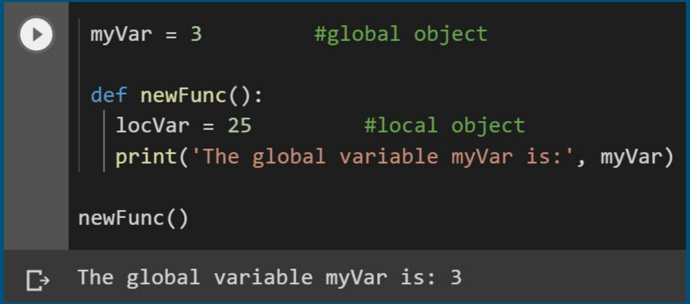
```
myVar = 3           #global object

def newFunc():
  locVar = 25          #local object
```

Accessing global & local variables



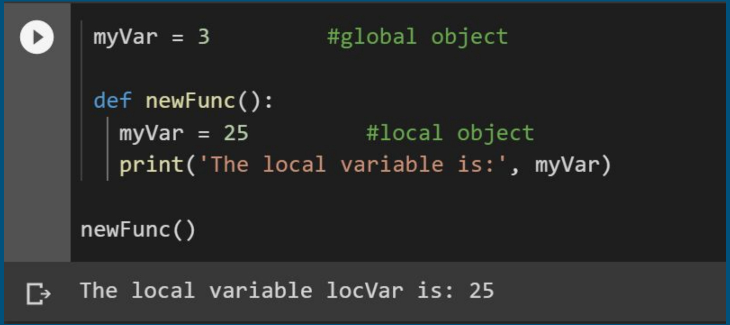
```
myVar = 3          #global object

def newFunc():
  myVar = 25          #local object

print('The global variable is:', myVar)
```

```
The global variable is: 3
```



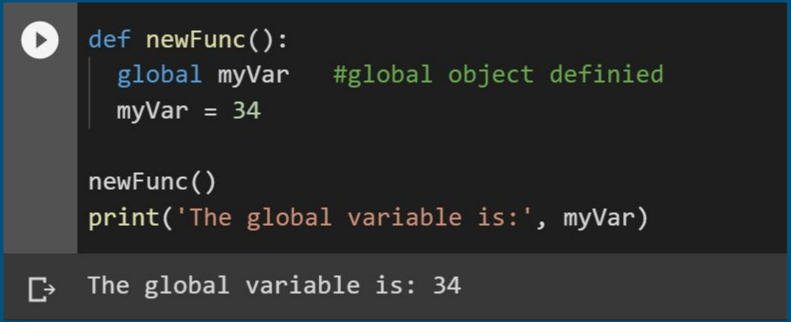
```
 myVar = 3          #global object

 def newFunc():
   locVar = 25          #local object
   print('The global variable myVar is:', myVar)

newFunc()
```

```
The global variable myVar is: 3
```



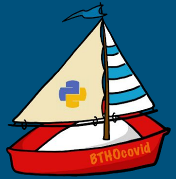
```
 myVar = 3          #global object

 def newFunc():
   myVar = 25          #local object
   print('The local variable is:', myVar)

newFunc()
```

```
The local variable locVar is: 25
```

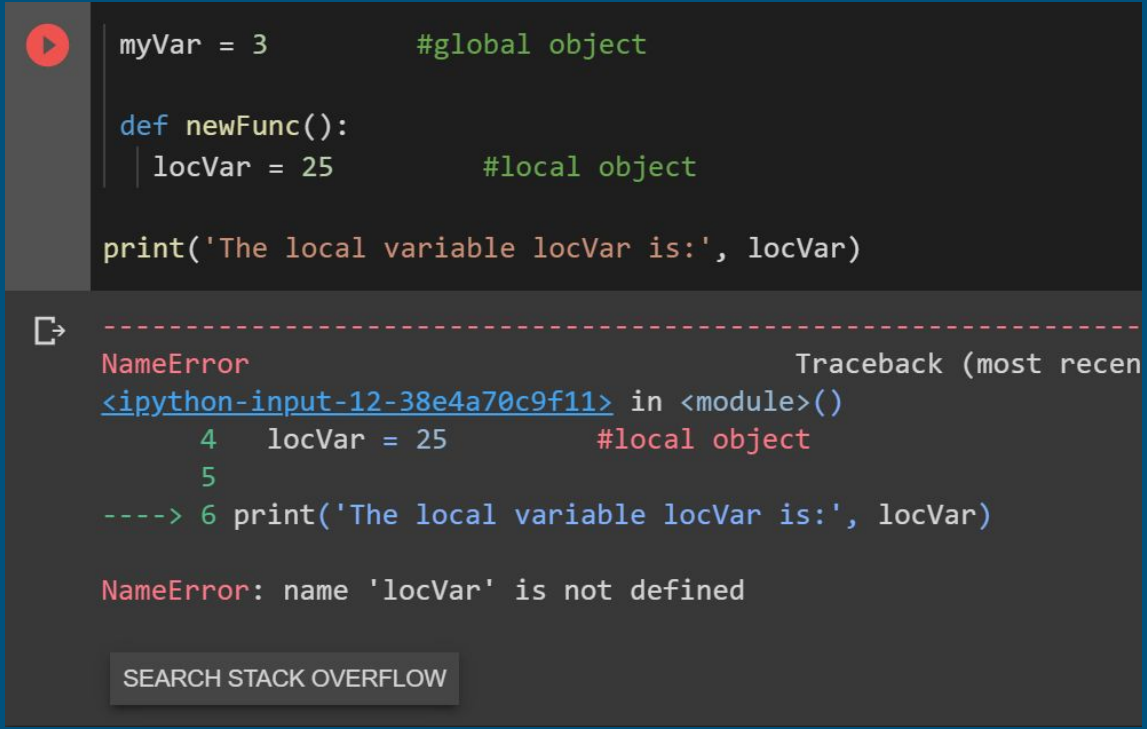


```
def newFunc():
  global myVar    #global object definied
  myVar = 34

newFunc()
print('The global variable is:', myVar)
```

```
The global variable is: 34
```



Accessing local object outside its scope

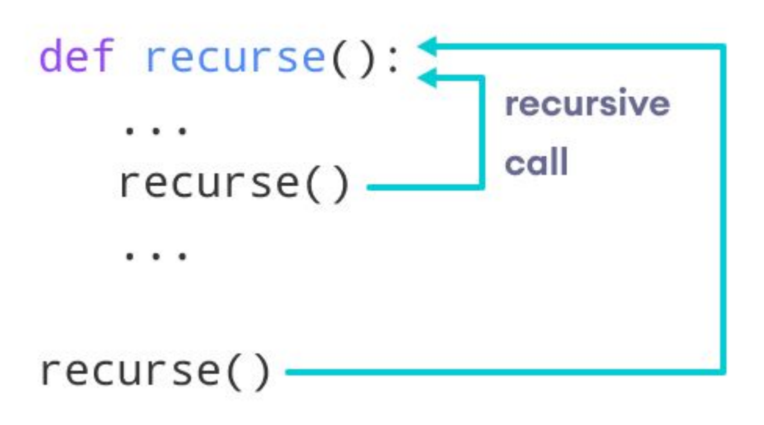


```
myVar = 3          #global object

def newFunc():
  locVar = 25          #local object

print('The local variable locVar is:', locVar)
```

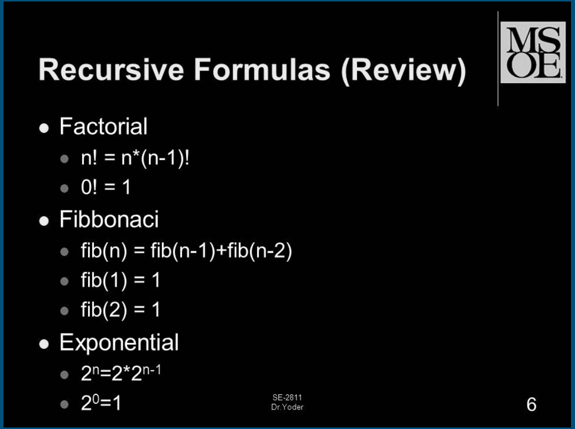
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recen
<ipython-input-12-38e4a70c9f11> in <module>()
      4   locVar = 25          #local object
      5 
----> 6 print('The local variable locVar is:', locVar)

NameError: name 'locVar' is not defined
```

Recursive Function Call

- We already know that a function can call other functions
- It is even possible for the function to call itself
- These types of functions are called recursive functions



```
def recurse():
    ...
    recurse()
    ...

recurse()
```



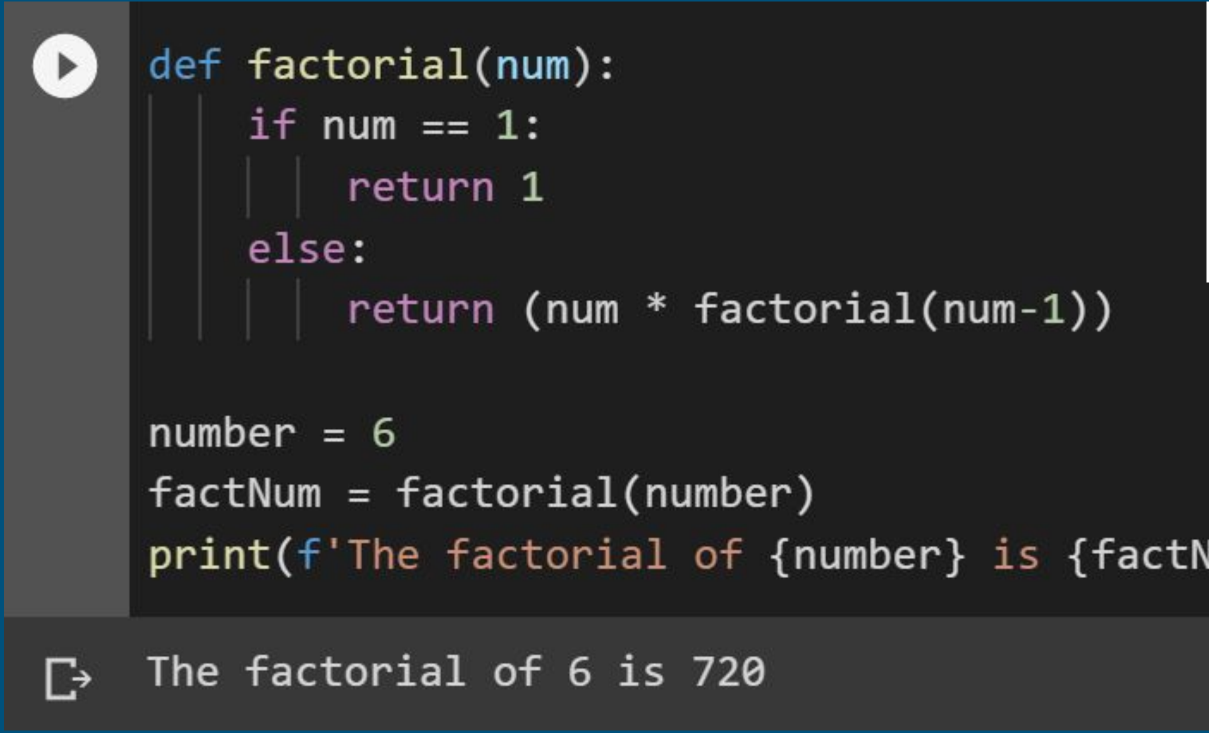
Recursive Formulas (Review)

- Factorial
  - n! = n*(n-1)!
  - 0! = 1
- Fibbonaci
  - fib(n) = fib(n-1)+fib(n-2)
  - fib(1) = 1
  - fib(2) = 1
- Exponential
  - $2^n=2*2^{n-1}$
  - $2^0=1$

SE-2811
Dr.Yoder

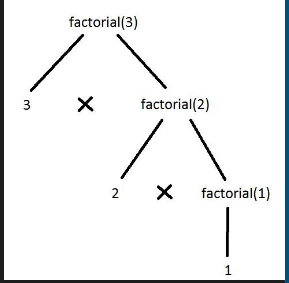
6

Example of recursive function

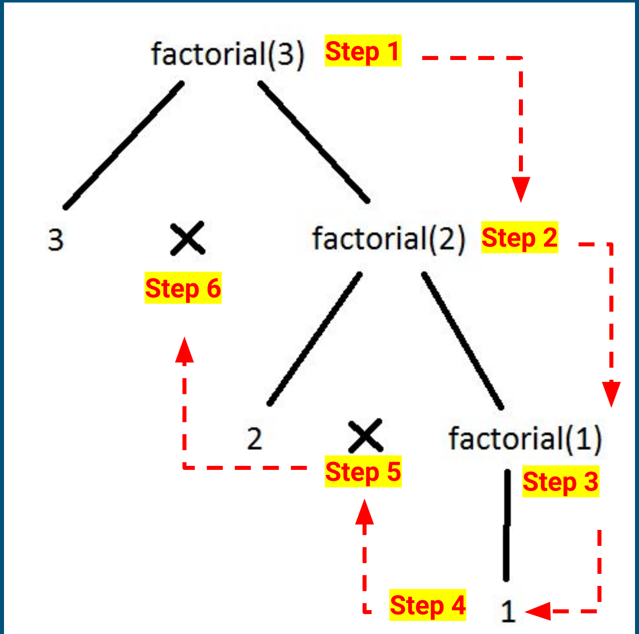


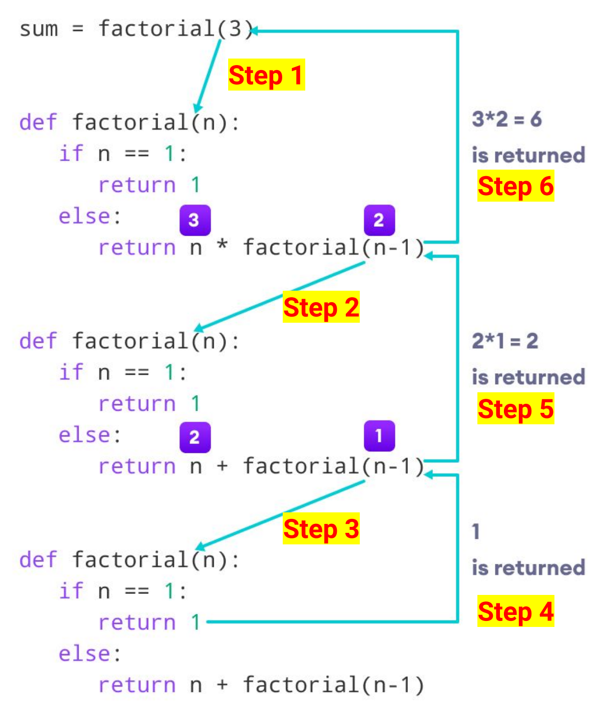
```python
def factorial(num):
    if num == 1:
        return 1
    else:
        return (num * factorial(num-1))

number = 6
factNum = factorial(number)
print(f'The factorial of {number} is {factNum}')
```

```
The factorial of 6 is 720
```



Visualizing recursive function call





```
sum = factorial(3)

def factorial(n):
   if n == 1:
      return 1
   else:
      return n * factorial(n-1)

def factorial(n):
   if n == 1:
      return 1
   else:
      return n + factorial(n-1)

def factorial(n):
   if n == 1:
      return 1
   else:
      return n + factorial(n-1)
```

Parameters vs Arguments

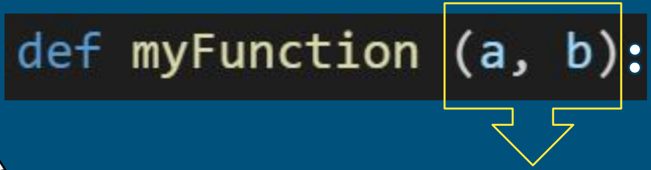
A function takes in some variables, does computation on them and then returns a value

These variables are called Parameters defined within ( ) of function

When a function is called from within the program, you supply some variables

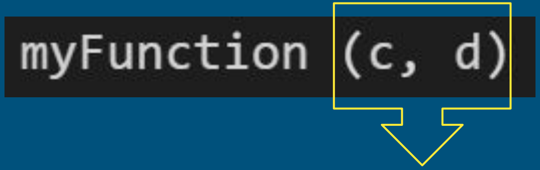
These supplied variables are called Arguments

These arguments do NOT necessarily have the same name as parameters



```
def myFunction (a, b):
```

parameters

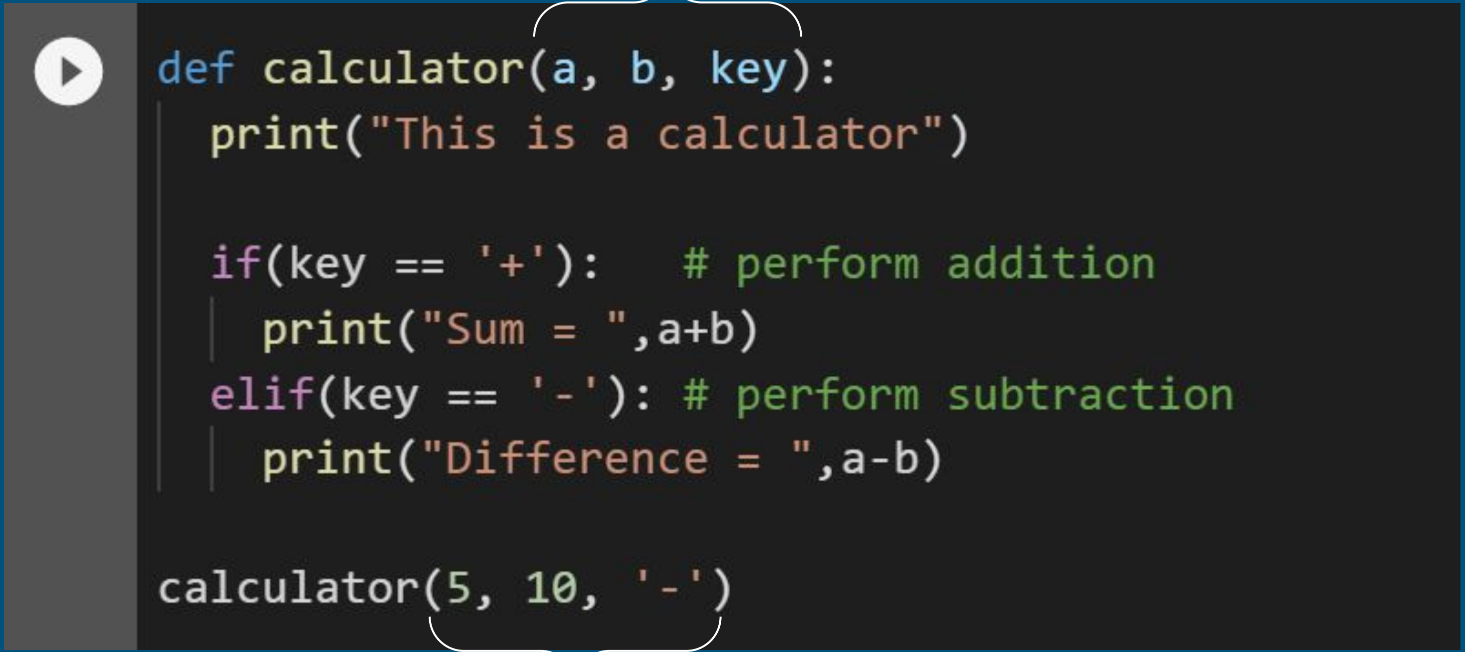


```
myFunction (c, d)
```

arguments

Examples

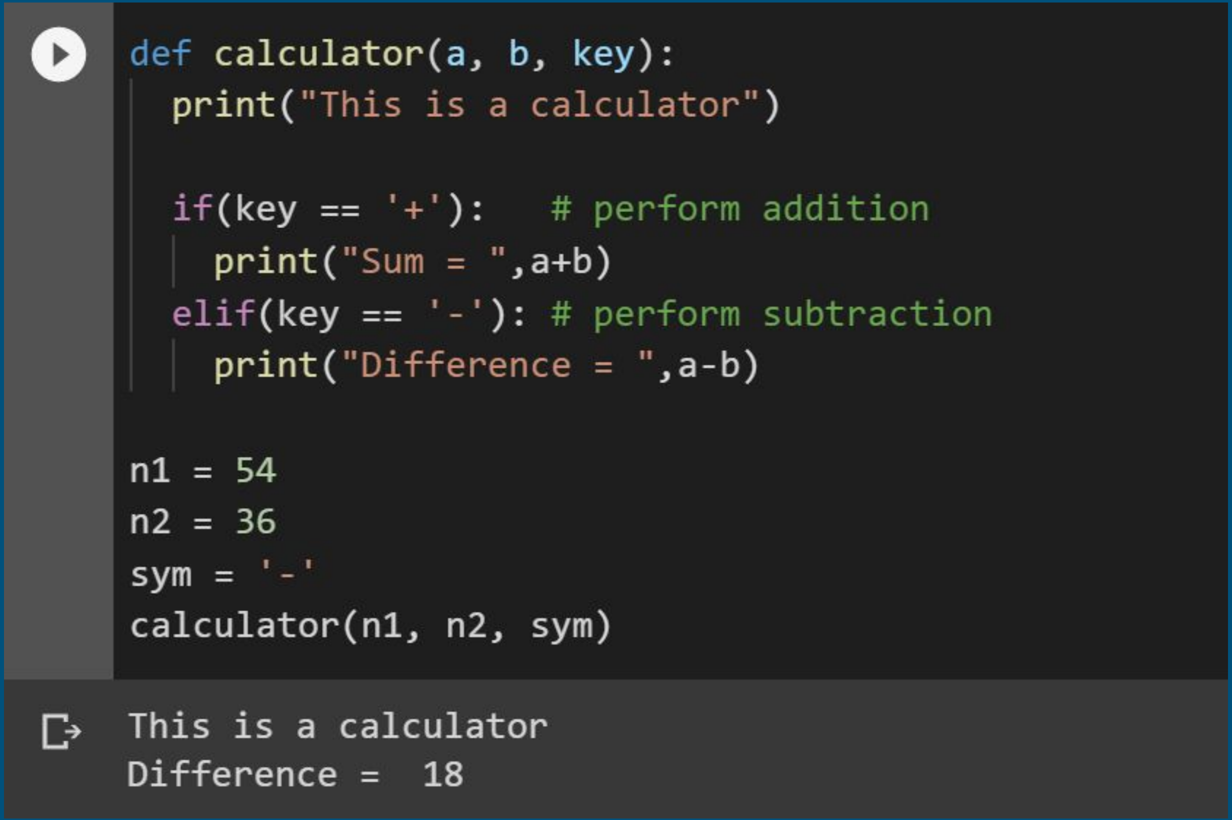
Parameters



```python
def calculator(a, b, key):
  print("This is a calculator")

  if(key == '+'):   # perform addition
    print("Sum = ",a+b)
  elif(key == '-'): # perform subtraction
    print("Difference = ",a-b)

calculator(5, 10, '-')
```

Arguments

Example - passing variables as arguments



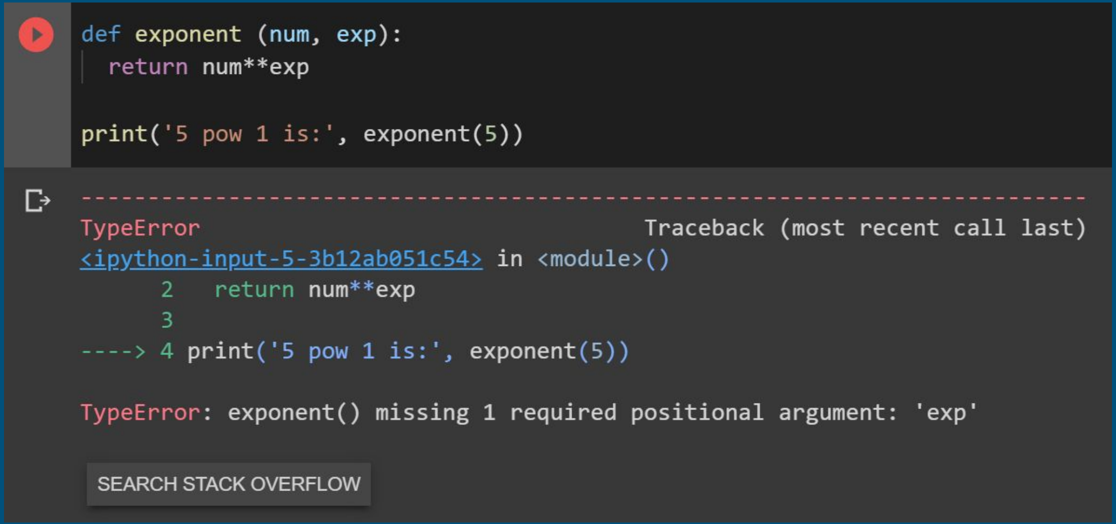
```python
def calculator(a, b, key):
  print("This is a calculator")

  if(key == '+'):   # perform addition
    print("Sum = ",a+b)
  elif(key == '-'): # perform subtraction
    print("Difference = ",a-b)

n1 = 54
n2 = 36
sym = '-'
calculator(n1, n2, sym)
```

```
This is a calculator
Difference =  18
```

Default argument

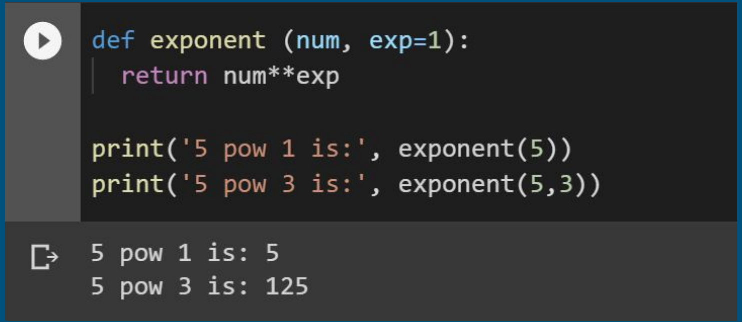
If a function defined with parameters is called without any arguments, it will throw an error

We can assign some default values to parameters. In case the function is called without arguments these **default values** gets assigned



```
def exponent (num, exp):
  return num**exp

print('5 pow 1 is:', exponent(5))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-5-3b12ab051c54> in <module>()
      2   return num**exp
      3
----> 4 print('5 pow 1 is:', exponent(5))

TypeError: exponent() missing 1 required positional argument: 'exp'
```



```
def exponent (num, exp=1):
  return num**exp

print('5 pow 1 is:', exponent(5))
print('5 pow 3 is:', exponent(5,3))
```

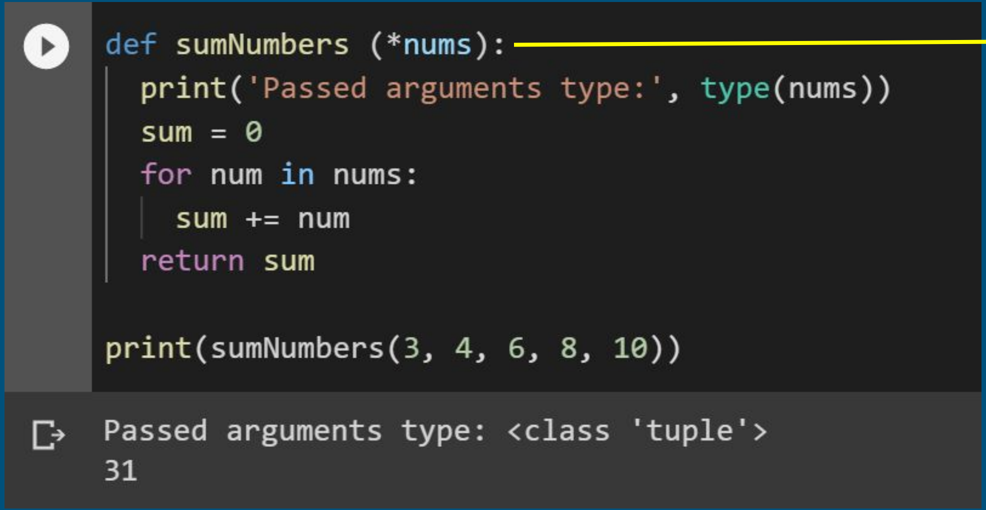
```
5 pow 1 is: 5
5 pow 3 is: 125
```

*args

Sometimes we do not know how many parameter needs to be passed to a function. In that case we can specify *args

The function can then accept multiple parameters when called

These parameters are stored as **Tuple** inside the function



```python
def sumNumbers (*nums):
  print('Passed arguments type:', type(nums))
  sum = 0
  for num in nums:
    sum += num
  return sum

print(sumNumbers(3, 4, 6, 8, 10))
```

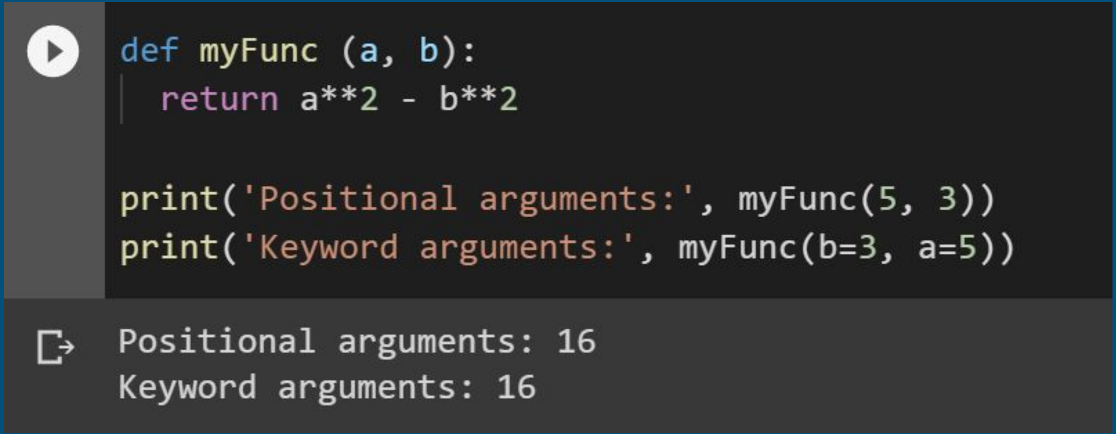
```
Passed arguments type: <class 'tuple'>
31
```

Any name

keyword arguments

Generally, the arguments from caller are assigned to the corresponding parameters in function definition in the same order

But if you want to assign the arguments from caller to the parameters in function definition without worrying about the order, we use keyword argument notation



```
def myFunc (a, b):
  return a**2 - b**2

print('Positional arguments:', myFunc(5, 3))
print('Keyword arguments:', myFunc(b=3, a=5))
```

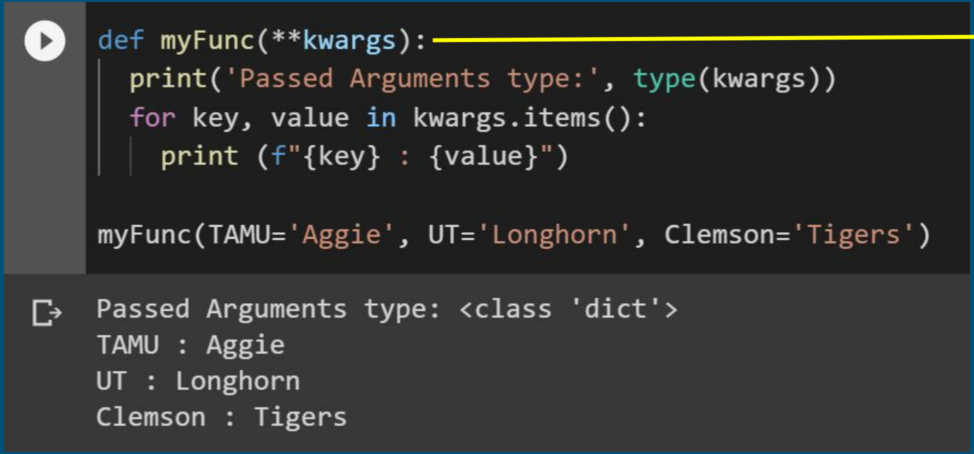
```
Positional arguments: 16
Keyword arguments: 16
```

**kwargs

**kwargs in functions definition is used to pass a keyword argument list of variable length

Think of it as *args but now each argument with its own name.

kwargs are stored in the function as a dictionary



```
def myFunc(**kwargs):
  print('Passed Arguments type:', type(kwargs))
  for key, value in kwargs.items():
    print (f"{key} : {value}")

myFunc(TAMU='Aggie', UT='Longhorn', Clemson='Tigers')
```

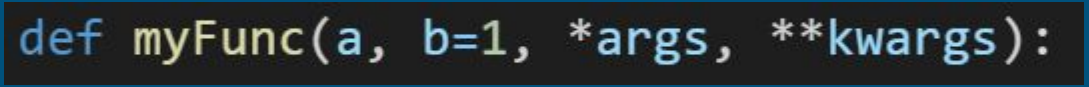
```
Passed Arguments type: <class 'dict'>
TAMU : Aggie
UT : Longhorn
Clemson : Tigers
```

Any name

Ordering Parameters / Arguments in a Function

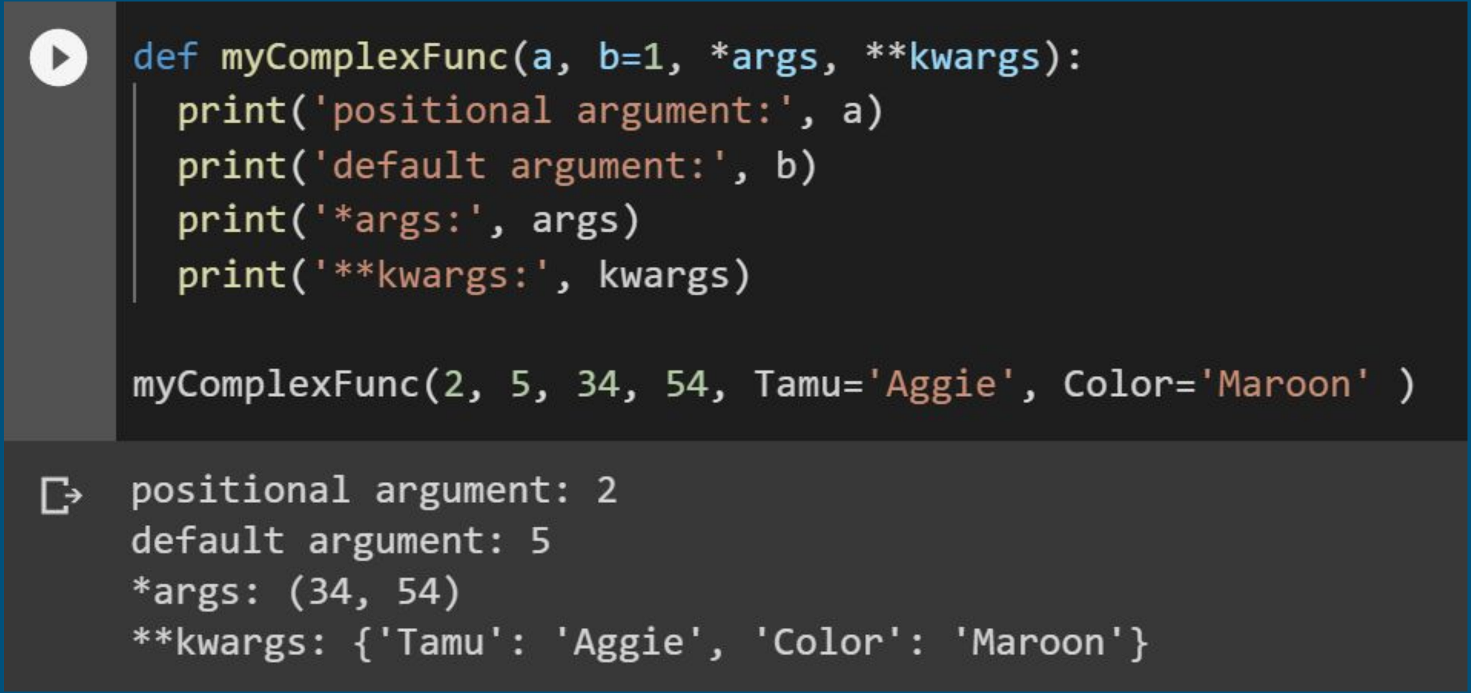
When you have multiple type of parameters in your function, there is an order you need to follow:

1. Required parameters/ positional parameters
2. Default parameters
3. *args parameters
4. **kwargs parameters



```
def myFunc(a, b=1, *args, **kwargs):
```

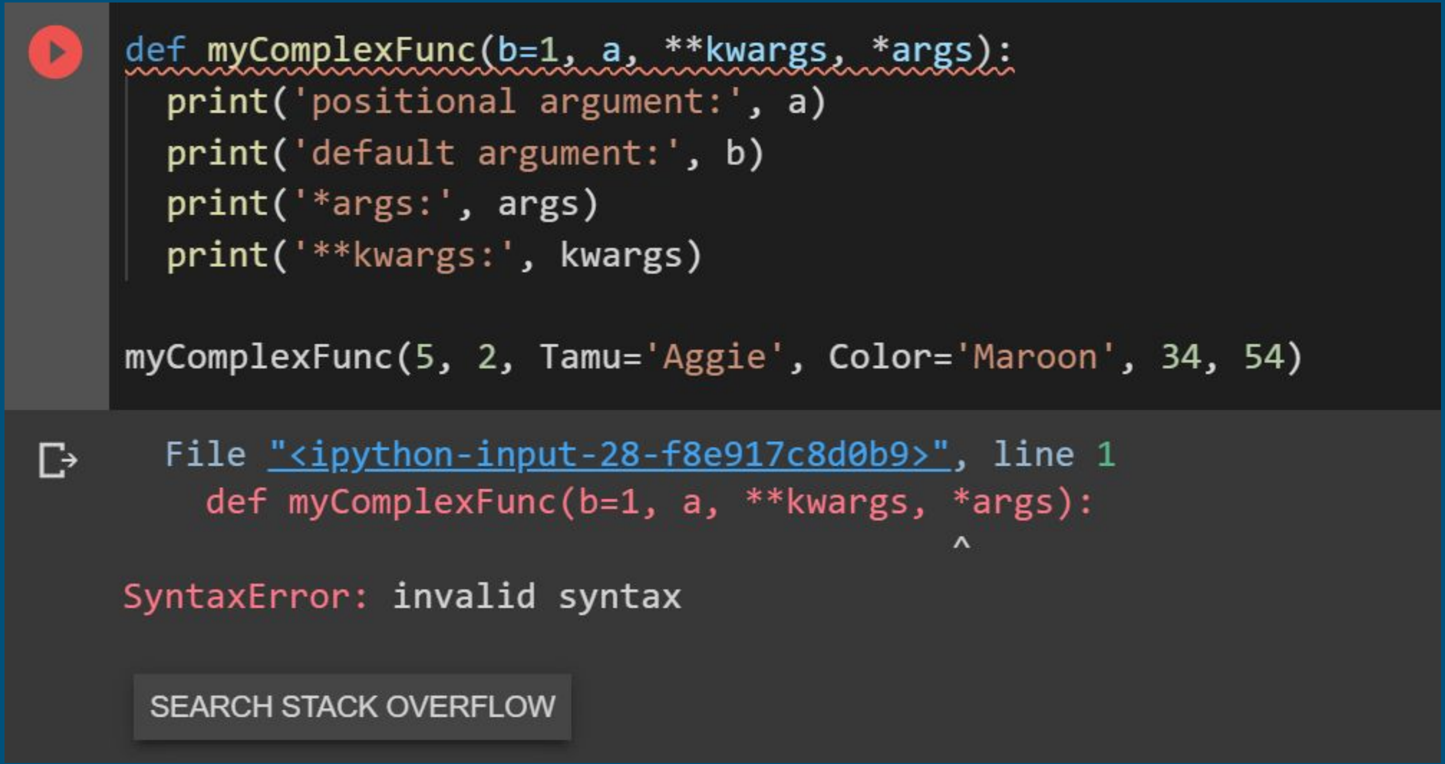
Example



```python
def myComplexFunc(a, b=1, *args, **kwargs):
  print('positional argument:', a)
  print('default argument:', b)
  print('*args:', args)
  print('**kwargs:', kwargs)

myComplexFunc(2, 5, 34, 54, Tamu='Aggie', Color='Maroon' )
```

```
positional argument: 2
default argument: 5
*args: (34, 54)
**kwargs: {'Tamu': 'Aggie', 'Color': 'Maroon'}
```

Incorrect order- Example



```python
def myComplexFunc(b=1, a, **kwargs, *args):
  print('positional argument:', a)
  print('default argument:', b)
  print('*args:', args)
  print('**kwargs:', kwargs)

myComplexFunc(5, 2, Tamu='Aggie', Color='Maroon', 34, 54)
```

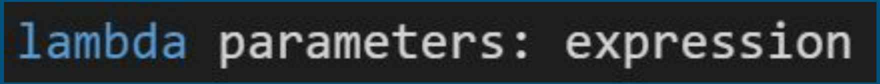
```
  File "<ipython-input-28-f8e917c8d0b9>", line 1
    def myComplexFunc(b=1, a, **kwargs, *args):
                                        ^
SyntaxError: invalid syntax
```

Lambda function - anonymous function

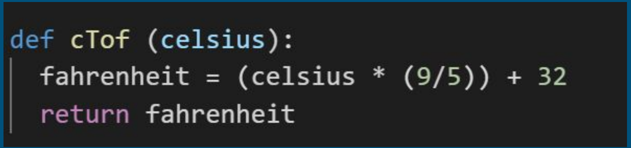
- When a function is simple enough to be written in a single line (meaning it has only one expression), it can be written using **lambda** notation

- This reduces amount of code and enhances code readability

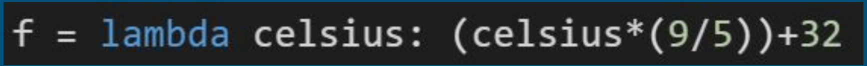
- No need for def to define a function



```
lambda parameters: expression
```



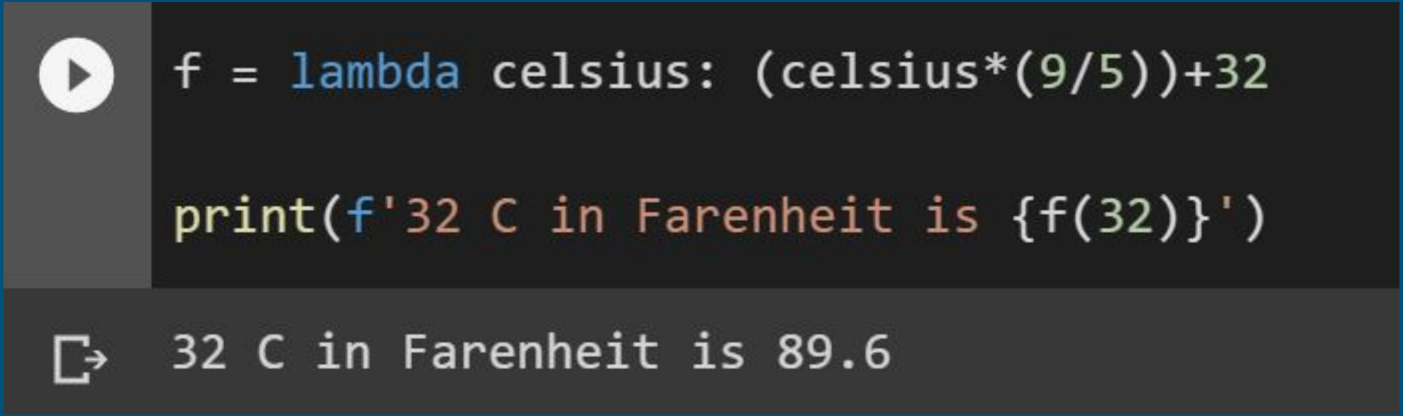
```
def cTof (celsius):
  fahrenheit = (celsius * (9/5)) + 32
  return fahrenheit
```



```
f = lambda celsius: (celsius*(9/5))+32
```

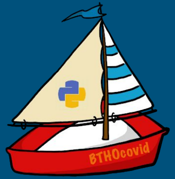
lambda function returns a function object

Example



```
f = lambda celsius: (celsius*(9/5))+32

print(f'32 C in Farenheit is {f(32)}')
```

```
32 C in Farenheit is 89.6
```



Next Lecture

File I/O & Python Scripts

Sat, May 09 (11 am-12 noon CDT)