SNS COLLEGE OF ENGINEERING
Kurumbapalayam (Po), Coimbatore – 641 107

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

19IT103 – COMPUTATIONAL THINKING AND PYTHON PROGRAMMING

- A readable, dynamic, pleasant, flexible, fast and powerful language

Recap

- Values present in the function calling statement are called arguments
- Variables used in the function header are called parameters
- Required, keyword, default and variable-length are types of arguments
- Variable can be created with local and global scopes
- Global keyword creates a global variable inside a block

Agenda

- Functions composition and Lambda functions
- Recursion

Functions composition

- Function composition is a way of combining functions such that the result of each function is passed as the argument of the next function.
- For example, the composition of two functions f and g is denoted f(g(x)).
- x is the argument of g, the result of g is passed as the argument of f and the result of the composition is the result of f.
- Function composition is achieved through lambda functions

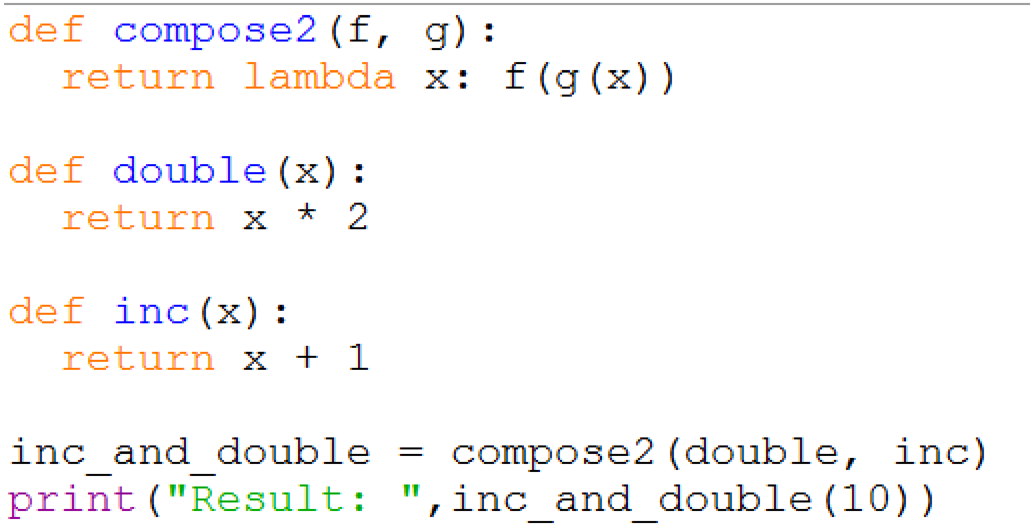
Functions composition

- Lambda functions are called anonymous because they are not declared in the standard manner by using the def keyword.
- You can use the lambda keyword to create small anonymous functions.
- Lambda can take any number of arguments but return just one value in the form of an expression. They cannot contain commands or multiple expressions.
- An anonymous function cannot be a direct call to print because lambda requires an expression

Functions composition

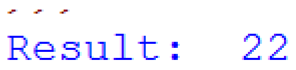
- For example, compose2 is a function that takes two functions as arguments (f and g) and returns a function representing their composition

Example:



```
def compose2(f, g):
  return lambda x: f(g(x))

def double(x):
  return x * 2

def inc(x):
  return x + 1

inc_and_double = compose2(double, inc)
print("Result: ",inc_and_double(10))
```

Output:

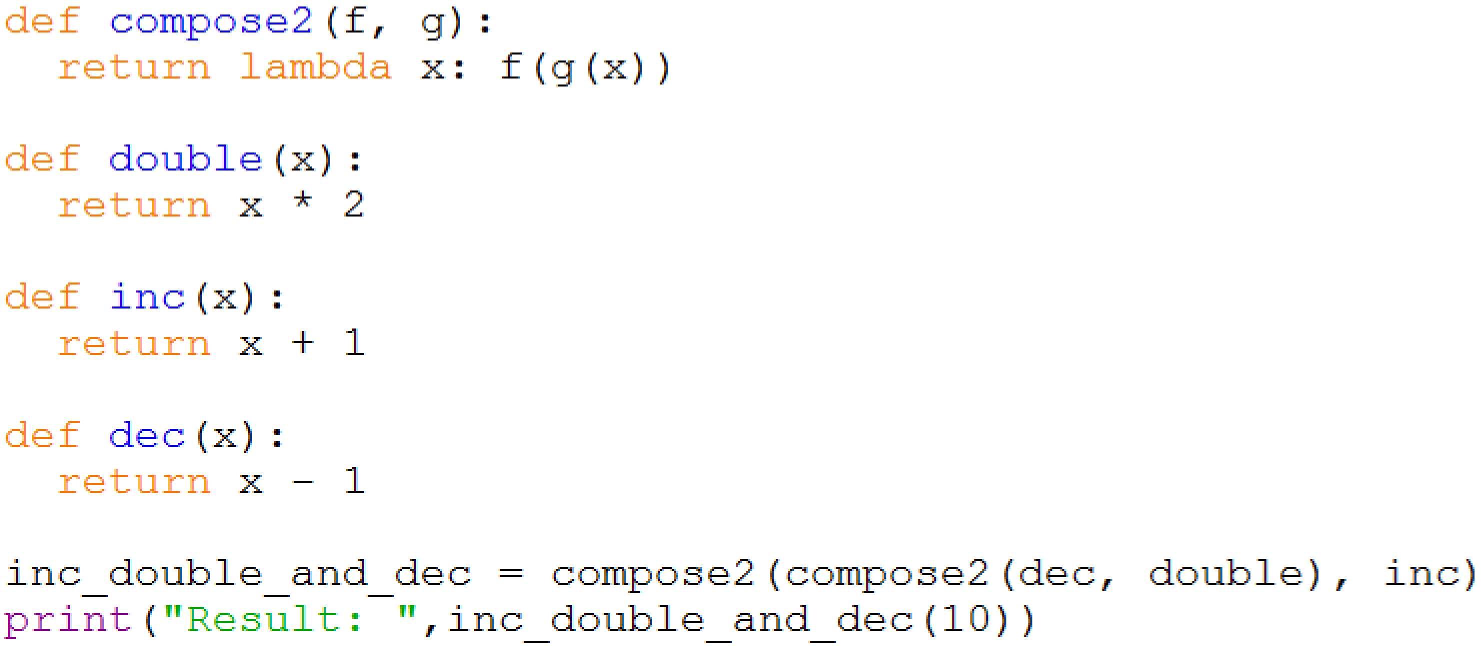


```
Result:  22
```

Composing *n* Functions

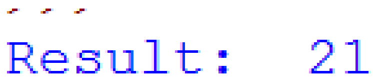
- It would be interesting to generalize the concept to accept *n* functions

Example:



```
def compose2(f, g):
   return lambda x: f(g(x))

def double(x):
   return x * 2

def inc(x):
   return x + 1

def dec(x):
   return x - 1

inc_double_and_dec = compose2(compose2(dec, double), inc)
print("Result: ",inc_double_and_dec(10))
```

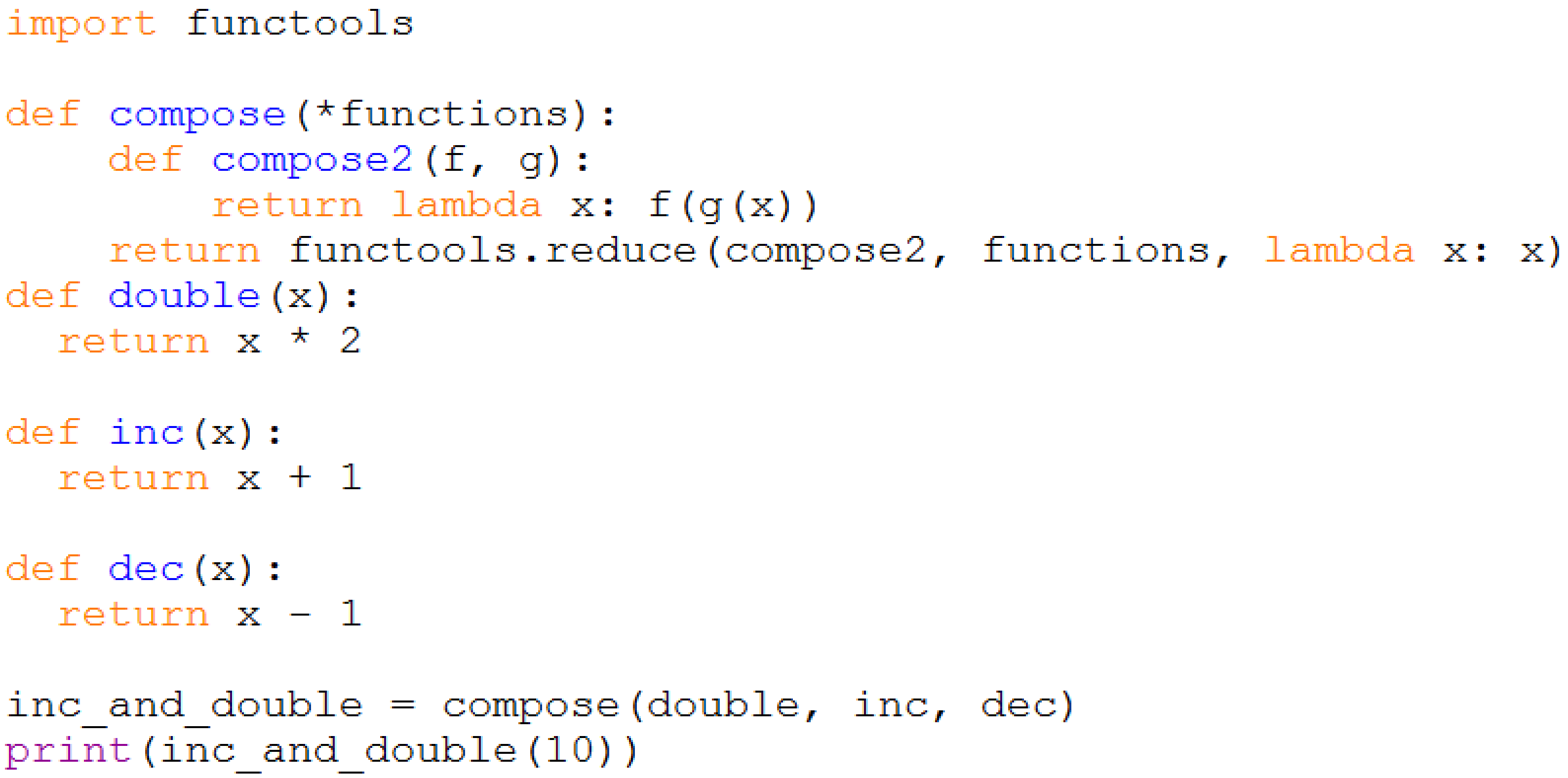
Output:



```
Result:  21
```

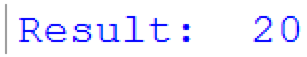
Composing *n* Functions using *“functools”*

Example:



```
import functools

def compose(*functions):
    def compose2(f, g):
        return lambda x: f(g(x))
    return functools.reduce(compose2, functions, lambda x: x)
def double(x):
  return x * 2

def inc(x):
  return x + 1

def dec(x):
  return x - 1

inc_and_double = compose(double, inc, dec)
print(inc_and_double(10))
```

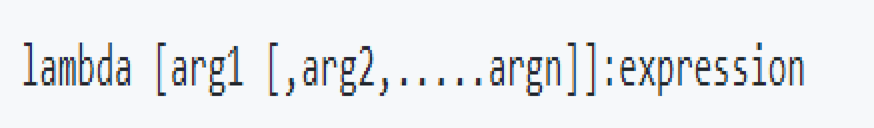
Output:



```
Result:  20
```

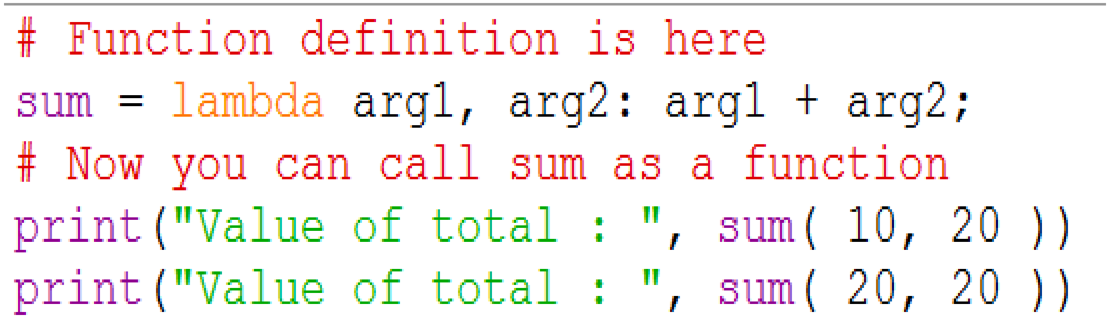
Functions composition

Syntax



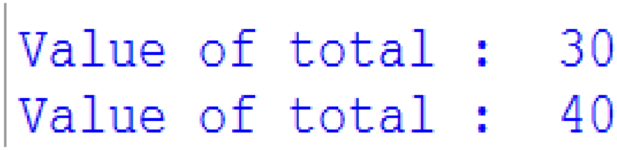
```
lambda [arg1 [,arg2,.....argn]]:expression
```

Example:



```python
# Function definition is here
sum = lambda arg1, arg2: arg1 + arg2;
# Now you can call sum as a function
print("Value of total : ", sum( 10, 20 ))
print("Value of total : ", sum( 20, 20 ))
```

Output:



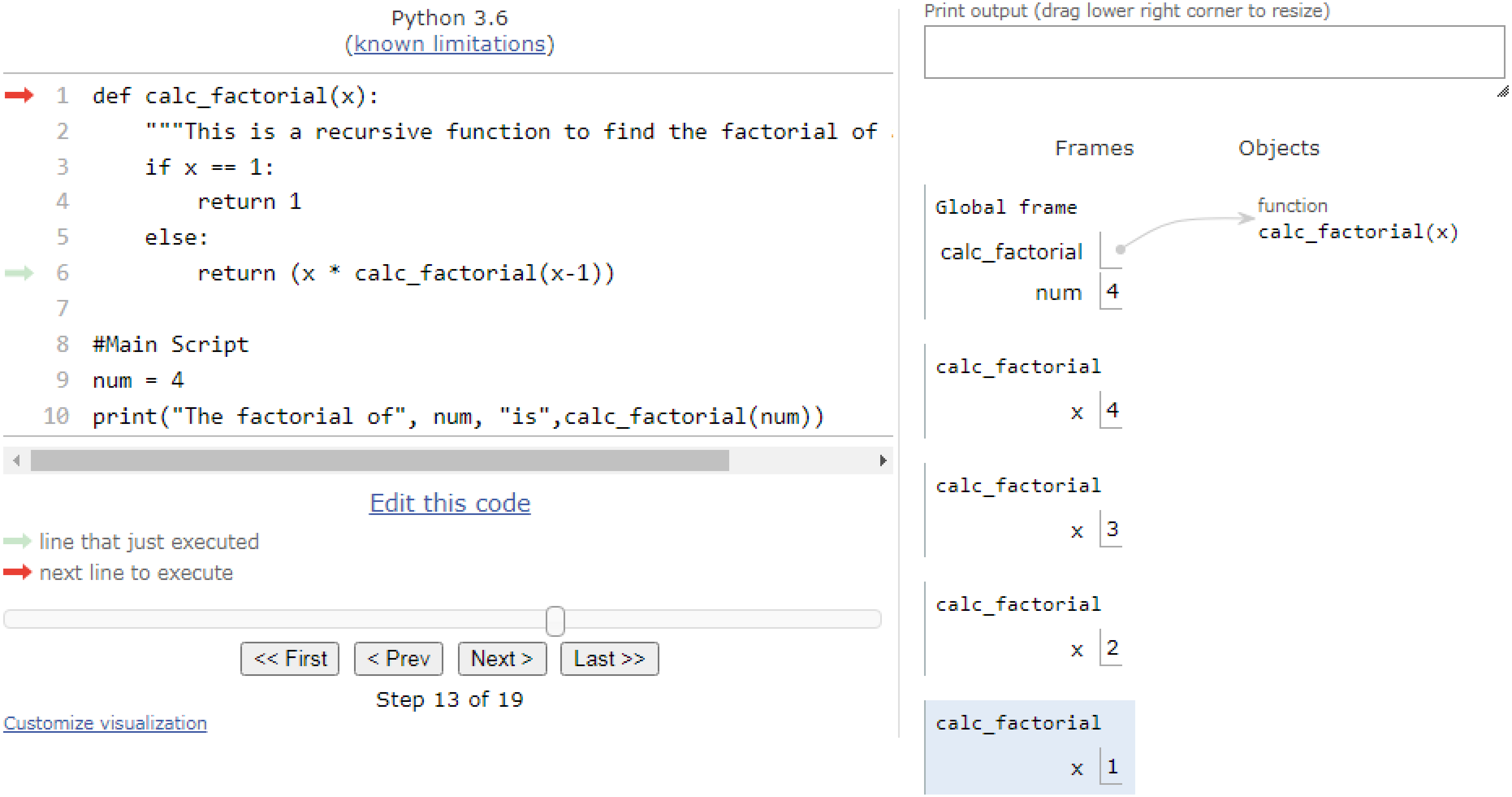
```
Value of total :  30
Value of total :  40
```

Recursion

- Recursion is the process calling a function by itself
- For example, to find the factorial of an integer can be written as recursive function.
- Factorial of a number is the product of all the integers from 1 to that number.
- For example, the factorial of 6 (denoted as 6!) is 12345*6 = 720.

Recursion

Example:



Python 3.6
(known limitations)

```
def calc_factorial(x):
    """This is a recursive function to find the factorial of
    if x == 1:
        return 1
    else:
        return (x * calc_factorial(x-1))

#Main Script
num = 4
print("The factorial of", num, "is",calc_factorial(num))
```

Edit this code

line that just executed
next line to execute

Step 13 of 19

Customize visualization

Print output (drag lower right corner to resize)

Frames

Global frame
calc_factorial
num 4

calc_factorial
x 4

calc_factorial
x 3

calc_factorial
x 2

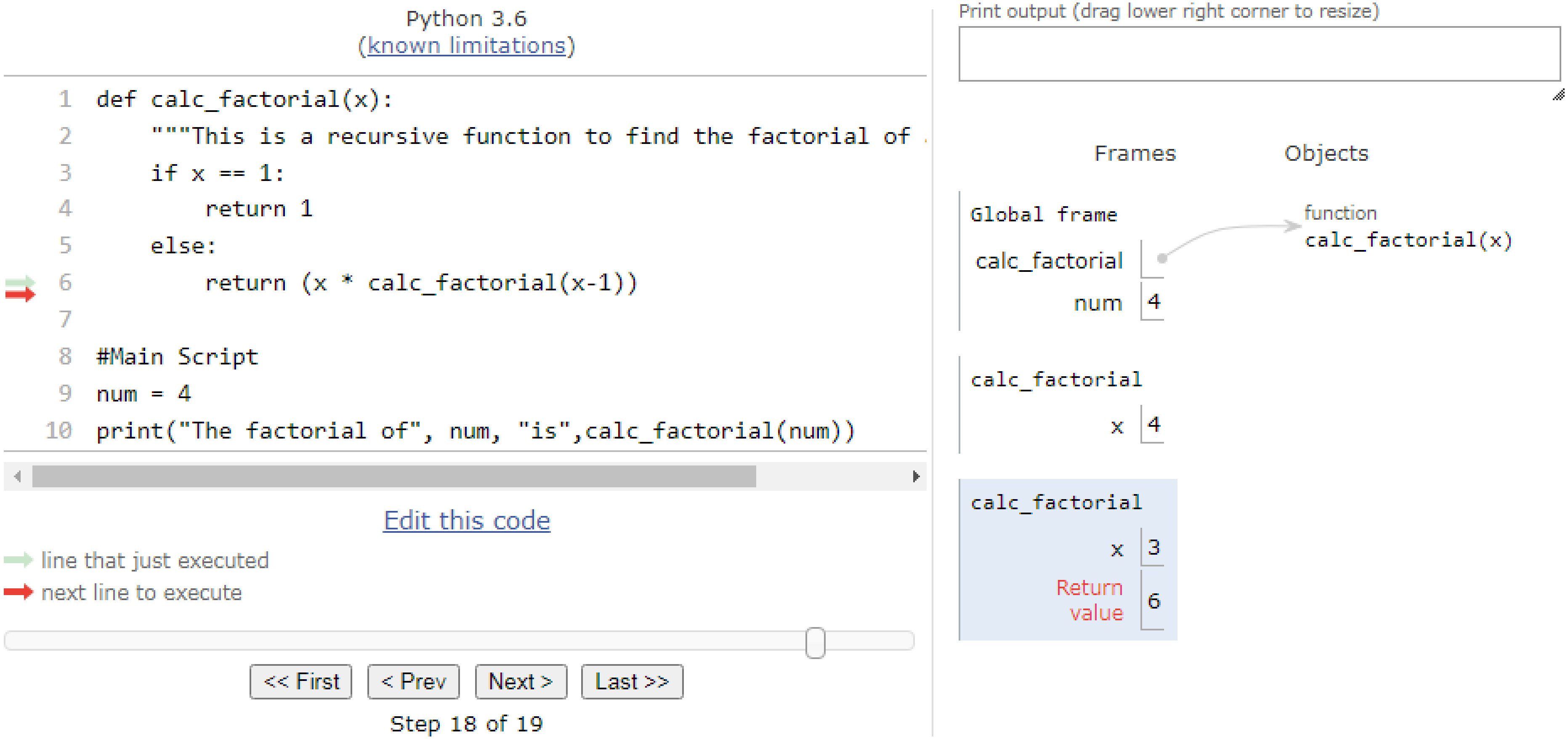
calc_factorial
x 1

Objects

function
calc_factorial(x)

Recursion

Output:



Python 3.6
(known limitations)

```
def calc_factorial(x):
    """This is a recursive function to find the factorial of
    if x == 1:
        return 1
    else:
        return (x * calc_factorial(x-1))

#Main Script
num = 4
print("The factorial of", num, "is",calc_factorial(num))
```

Edit this code

➡ line that just executed
➡ next line to execute

Step 18 of 19

Print output (drag lower right corner to resize)

Frames

Global frame
calc_factorial
num 4

calc_factorial
x 4

calc_factorial
x 3
Return value 6

Objects

function
calc_factorial(x)

Recursion

- Our recursion ends when the number reduces to 1. This is called the base condition.
- Every recursive function must have a base condition that stops the recursion or else the function calls itself infinitely.

Recursion

Advantages of recursion

- Recursive functions make the code look clean and elegant.
- A complex task can be broken down into simpler sub-problems using recursion.
- Sequence generation is easier with recursion than using some nested iteration.

Recursion

Disadvantages of recursion

- Sometimes the logic behind recursion is hard to follow through.
- Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
- Recursive functions are hard to debug.

Summary

- Function composition is a way of combining functions
- Recursion is the process calling a function by itself
- Function composition is achieved through lambda functions
- Lambda functions are called anonymous because they are not declared in the standard manner by using the def keyword

THANK YOU