SNS COLLEGE OF ENGINEERING
Kurumbapalayam (Po), Coimbatore – 641 107
An Autonomous Institution
Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF CSE

19IT103 – COMPUTATIONAL THINKING AND PYTHON PROGRAMMING

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

Recap

- “break” statement is used terminate the loop in between the iterations
- “continue” statement is used to skip an iteration
- “pass” statement acts as a placeholders for future code
- Python Functions is a block of related statements designed to perform a computational, logical, or evaluative task.
- Flow of execution is the order in which statements are executed

Agenda

- Functions
  - Arguments vs parameters
  - Types of arguments
- Fruitful functions
- Local and global scope

3.4 Functions

Arguments vs parameters

Arguments	Parameters
Arguments are used when a function is called	Parameters are used when function is to be defined
An argument is a value passed to a function when calling the function.	A parameter is a named entity in a function definition that specifies an argument that the function can accept.
Very often, there is a 1:1 mapping between arguments and...	...the parameters defined in the function.
Arguments can be constants, local variables or objects	Parameters cannot be a constant.
There are keyword arguments, and there are positional arguments. Anything that is not a keyword argument is a positional argument.	Keyword parameters and positional parameters

cont'd

3.4 Functions

Arguments vs parameters

The scope of the arguments are relevant only in the called function	The scope of the parameters have valid scope only within the function where they occur
Types of Arguments **Positional/Required**: Arguments without a name. **keyword**: Arguments with a name. **default**: a value provided in a function declaration that is automatically assigned. It is more precise to refer to this as "parameter with default value" **Variable-length arguments:** Variable length argument make function calls with arbitrary number of arguments	**Types of Parameters** **positional-or-keyword:** parameters in a function definition, with or without default values. **positional-only:** Only found in builtin/extension functions. **var-positional:** This is the *args. **keyword-only**: parameters that come after a * or *args, with or without default values. **var-keyword:** This is the **args

3.4 Functions

Types of Arguments

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

3.4 Functions

Types of Arguments

1. Required arguments

- Required arguments are the arguments passed to a function in correct positional order.
- The number of arguments in the function call should match exactly with the function definition.

3.4 Functions

Types of Arguments

1. Required arguments

Example:

```
#Function definition where str is the required argument
def display(str):
        print(str)

#main script
str="hello"
display(str)
```

Output:

```
hello
>>> |
```

3.4 Functions

Types of Arguments

2. Keyword arguments

- When keyword arguments in a function call, the caller identifies the arguments by the parameter name.
- This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters.

3.4 Functions

Types of Arguments

2. Keyword arguments

Example:

```
def display(val1,val2,val3):
        print("The string is:",val1)
        print("The integer is:",val2)
        print("The float is:",val3)

#Main script-Fucntion calling with keyword arguments
display(val3=58.62,val1="hello",val2=28)
```

Output:

```
The string is: hello
The integer is: 28
The float is: 58.62
```

3.4 Functions

Types of Arguments

3. Default arguments

- A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

3.4 Functions

Types of Arguments

3.Default arguments

Example:

```
def printinfo(name, age = 35 ):
        "This prints a passed info into this function"
        print("Name: ",name)
        print("Age ",age)
        return
#Calling printinfo function
printinfo(age=50, name="miki" )
printinfo(name="miki")
```

Output:

```
'''
Name:  miki
Age  50
Name:  miki
Age  35
```

3.4 Functions

Types of Arguments

4. Variable-length arguments

- Variable length argument make function calls with arbitrary number of arguments
- These arguments are called variable-length arguments and are not named in the function definition, unlike required and default arguments.
- **Syntax:**

```
def functionname([formal_args,] *var_args_tuple ):
        "function_docstring"
        function body
        return [expression]
```

3.4 Functions

Types of Arguments

4. Variable-length arguments

Example:

```
# Function definition is here
def printinfo( arg1, *vartuple ):
    "This prints a variable passed arguments"
    print ("Output is: ")
    print (arg1)
    for var in vartuple:
        print (var)
    return;

# Now you can call printinfo function
printinfo( 10 )
printinfo( 70, 60, 50 )
```

Output:

```
Output is:
10
Output is:
70
60
50
```

3.4 Functions – Fruitful functions

- The statement return [expression] exits a function, optionally passing back an expression to the caller.
- A return statement with no arguments is the same as return None.
- If a function returns some value then it is called as fruitful function
- Def: A function with a return value is called fruitful function

3.4 Functions –Fruitful functions

Example:

```
# Function definition is here
def sum( arg1, arg2 ):
        # Add both the parameters and return them."
        total = arg1 + arg2
        print("Inside the function : ", total)
        return total;

# Now you can call sum function
total = sum( 10, 20 );
print("Outside the function : ", total)
```

Output:

```
Inside the function :  30
Outside the function :  30
```

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:

```
# 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:

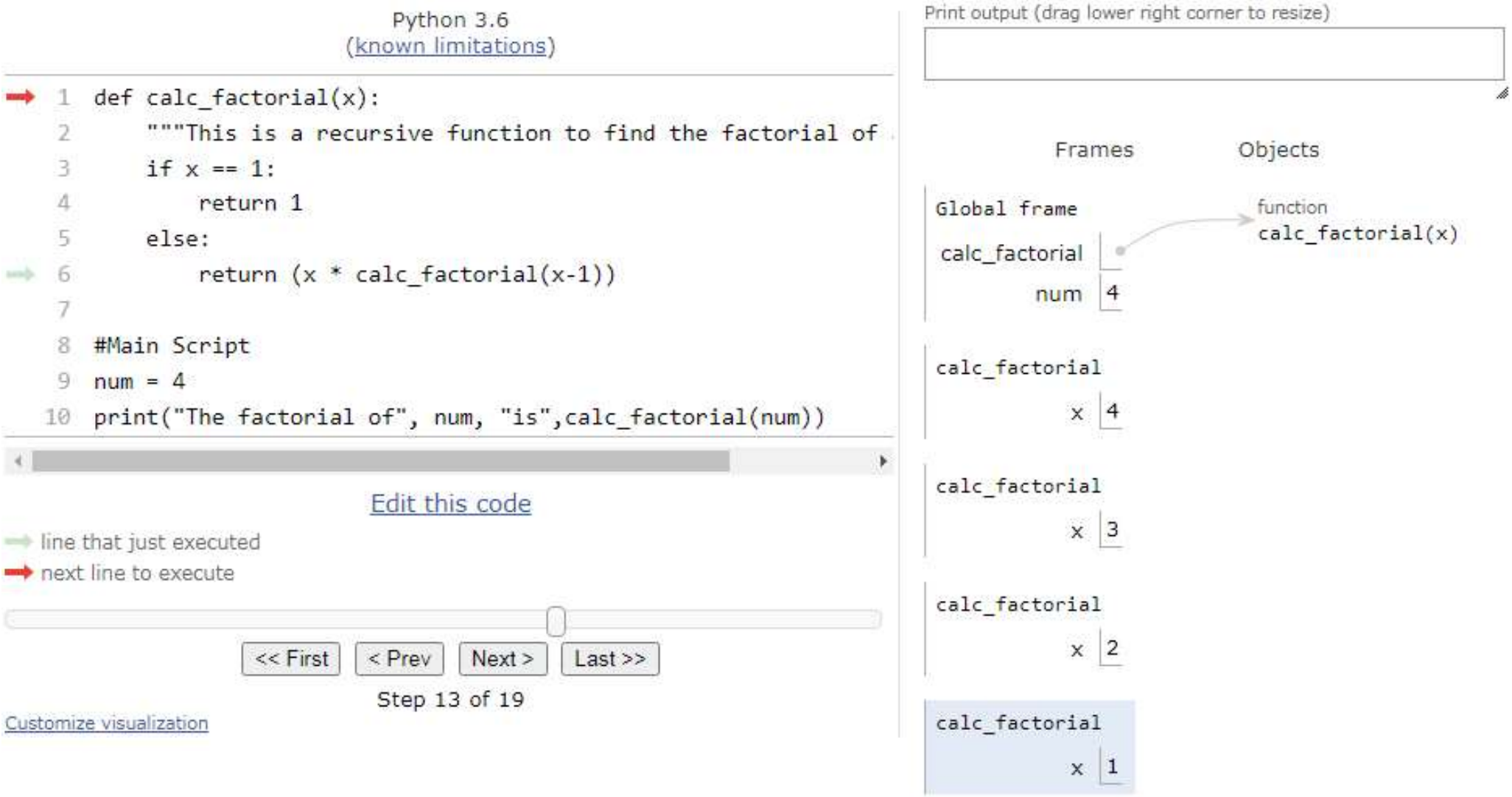
```
Value of total :  30
Value of total :  40
```

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.

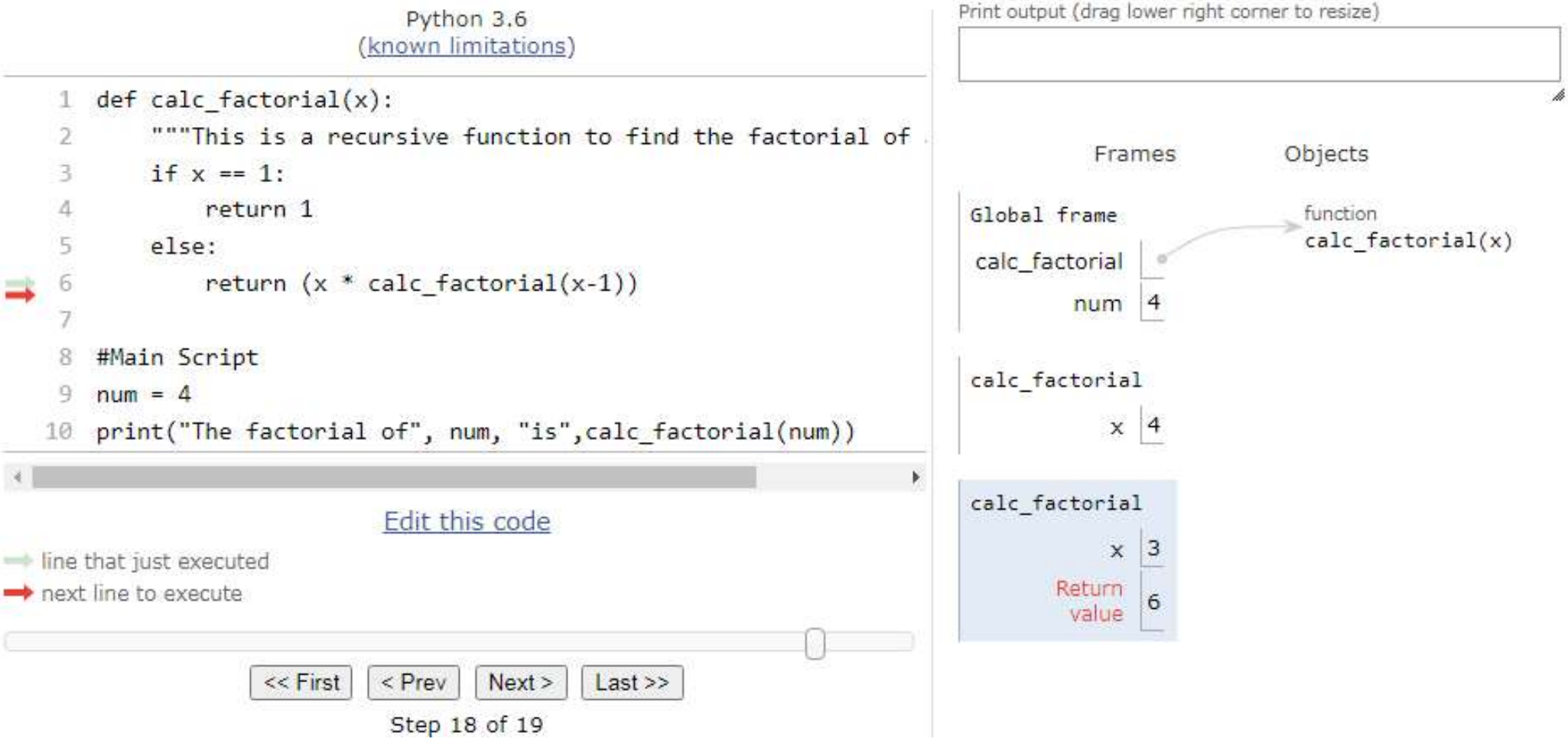
Recursion

Example:



Recursion

Output:



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