Module 5

This is a single, concatenated file, suitable for printing or saving as a PDF for offline viewing. Please note that some animations or images may not work.

Module 5 Study Guide and Deliverables

Theme: Functions in Detail

Readings:
- Chapter 6 (pp. 282-285), Chapter 14 (pp. 667-672), Chapter 5, Chapter 8, Chapter 15, and Chapter 16 (pp. 724-736)
- Module Lecture Notes

Topics: Exceptions, Introduction to Functions, Parameter Passing, Generators, Recursive Functions, Functional Programming

Assignments Assignment 5 due on Tuesday, April 20 at 6:00 PM ET

Assessments Quiz 5:
- Available Friday, April 16 at 6:00 AM ET
- Due on Tuesday, April 20 at 6:00 PM ET

Live Classrooms:
- Tuesday, April 13, 8:00 - 9:30 PM ET
- Thursday, April 15, 6:00 - 7:30 PM ET
- Facilitator Session: Friday, April 16, at 8:00 PM ET

Learning Objectives

At the end of this module, the learner is expected to be able to do the following:

- Distinguish interrupts and exceptions.
- Explain Python mechanisms to process exceptions.
- Define and use functions.
- Distinguish local and global scope.
- Describe role of mutability in parameter binding/passing.
- Distinguish *return* and *yield* statements.
- Describe the role of generators.
- Compare recursive and non-recursive functions.
- Use lambda functions andtheir use in functional programming.

Exception Handling

Exceptions and Interrupts

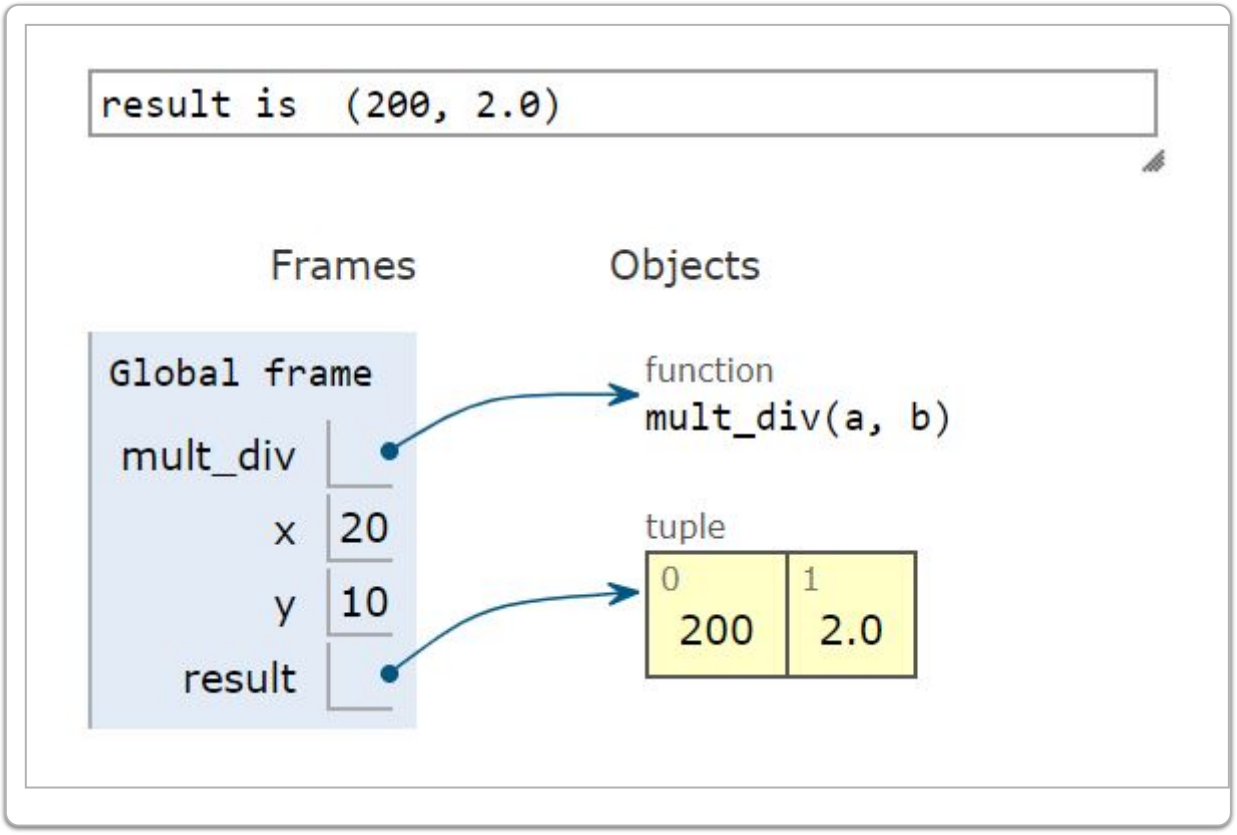
- Both change program flow
- Interrupts:
    a. caused by external events
    b. ex: network disruption
- Exceptions:
    a. caused by a program
    b. ex: division by zero
- Unhandled exceptions stop execution
- Mechanisms to "catch" and process exceptions

No Errors

```
def mult_div(a, b):
    mult_result = a * b
    div_result = a / b
```

```
    return mult_result, div_result

result = mult_div(20 , 10)
print('result is ', result)
```

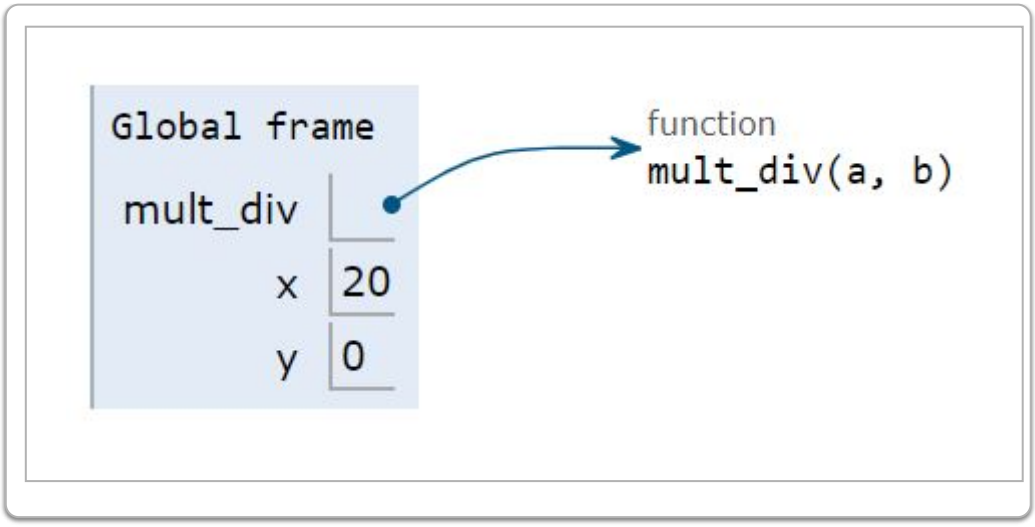


- no errors

An Exception Example

```
def mult_div(a, b):
    mult_result = a * b
    div_result = a / b
    return mult_result, div_result

result = mult_div(20, 10)
print('result is ', result)
```



- ZeroDivisionError

Raising Exceptions

```
def mult_div(a, b):
    if b == 0:
        raise Exception ('divide by zero!')
        mult_result, div_result = None, None
    else:
        mult_result = a * b
        div_result  = a / b
    return mult_result, div_result

result = mult_div(20 , 0)
```

- can define exceptions
- Exception: divide by zero!
- raising exceptions stops a program

Handling Exceptions

```
def mult_div(a, b):
    try:
        mult_result = a * b
        div_result  = a / b
    except Exception as e:
        print('Python error :', e)
        print('user-defined error: set to None')
        mult_result, div_result = None, None
    return mult_result, div_result
```

```
x = 20; y = 0
result = mult_div(x, y)
print('result is ', result)
```

```
Python error: division by zero
user-defined error: set to None
result is  (None, None)
```

Optional *finally* Clause

```
def mult_div(a, b):
    try:
        mult_result = a * b
        div_result = a / b
    except Exception as e:
        print('Python error :', e)
        print('user-defined error: set to None')
        mult_result, div_result = None, None
    finally :
            print('execution continues')
    return mult_result, div_result

print('mult_div(20 ,10) is',mult_div(20 ,10), '\n')
print('mult_div(20 , 0) is',mult_div(20 ,0))
```

```
execution continues
mult_div(20, 10) is (200, 2.0)

Python error: division by zero
user-defined error: set to None
execution continues
mult_div(20, 0) is (None, None)
```

Exception Examples

Name	Description

Exception	base class
ArithmeticError	errors in computation
ZeroDivisionError	division by zero
ImportError	import statement fails
IndexError	index not in sequence
KeyError	key not in dictionary
NamedError	identifier not found
SyntaxError	error in syntax
IndentationError	improper indentation

Multiple Exceptions

```
try:
    statement (s) - no errors
except exception_group_1 as error_1:
    statement (s) for error_1
    ----------------------------
    ----------------------------
except exception_group_n as error_n:
    statements for error_n
else:
    statement(s) for unknown error(s)
finally:
    final statement(s)
```

- handling multiple exceptions

Test Yourself: 5.1.01

Write a function *ratio_list*() to compute the ratio of first two elements in a list with the following criteria:

- The function must be capable to catch the following errors:
  a. IndexError
  b. ZeroDivisionError
- If an exception is generated, function should return *None*

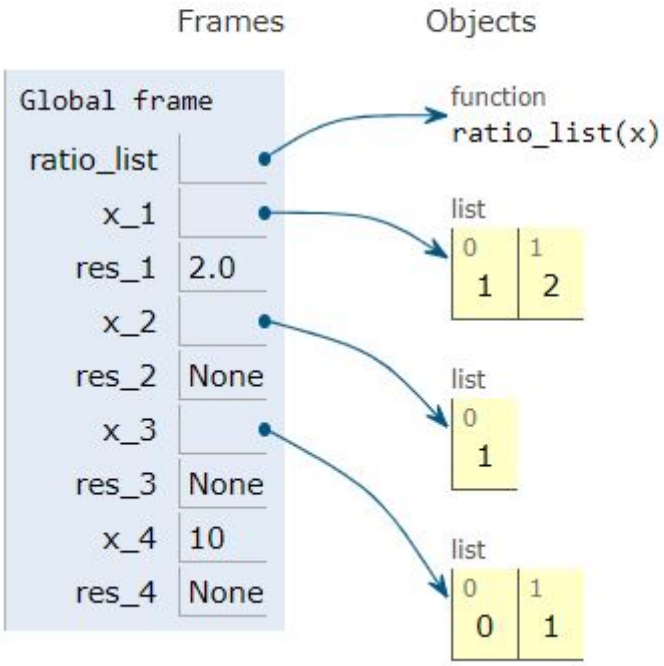
Solution:

```
def ratio_list(x):
    result = None
    try :
        result = x [1]/ x [0]
    except IndexError as e:
        print("input: ", x, "error: ", e)
    except ZeroDivisionError as e:
        print("input: ", x, "error: ", e)
    except :
        print("input: ", x, "some error")
    return result

x_1 = [1, 2]; res_1 = ratio_list(x_1)
print ("the result for ", x_1 , "is ", res_1)
x_2 = [1]; res_2 = ratio_list(x_2)
print ("the result for ", x_2 , "is ", res_2)

x_3 = [0, 1]; res_3 = ratio_list(x_2)
print ("the result for ", x_3 , "is ", res_3)
x_4 = 10; res_4 = ratio_list(x_3)
print ("the result for ", x_4 , "is ", res_4)
```

```
the result for  [1, 2]  is  2.0
input:  [1] error:  list index out of range
the result for  [1]  is  None
input:  [1] error:  list index out of range
the result for  [0, 1]  is  None
input:  [0, 1] error:  division by zero
the result for  10  is  None
```



Introduction to Functions

Functions

In this section, we will learn how to define and use functions and distinguish local and global scope.

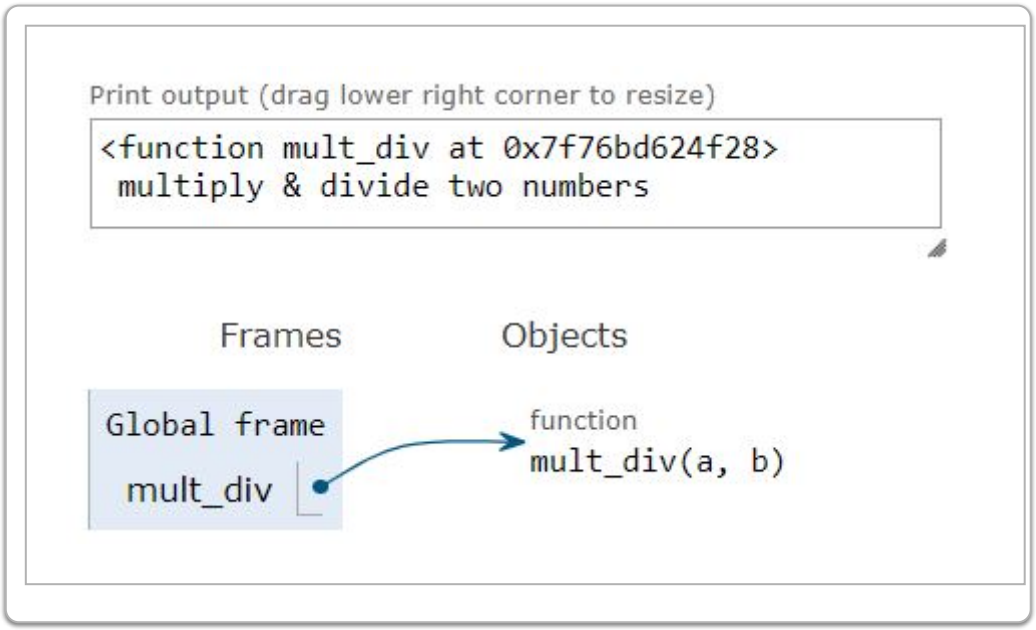
```
def mult_div(a, b):
    """ multiply & divide two numbers """
    result = a * b, a / b
    return result
```

- *def* keyword, name and parameters
- *doctring* – describes function
- statement(s) to compute
- optional *return* statement

Docstring

```
def mult_div(a, b):
    """ multiply & divide two numbers """
    result = a * b, a / b
    return result

print(mult_div)
print(mult_div.__doc__)
```

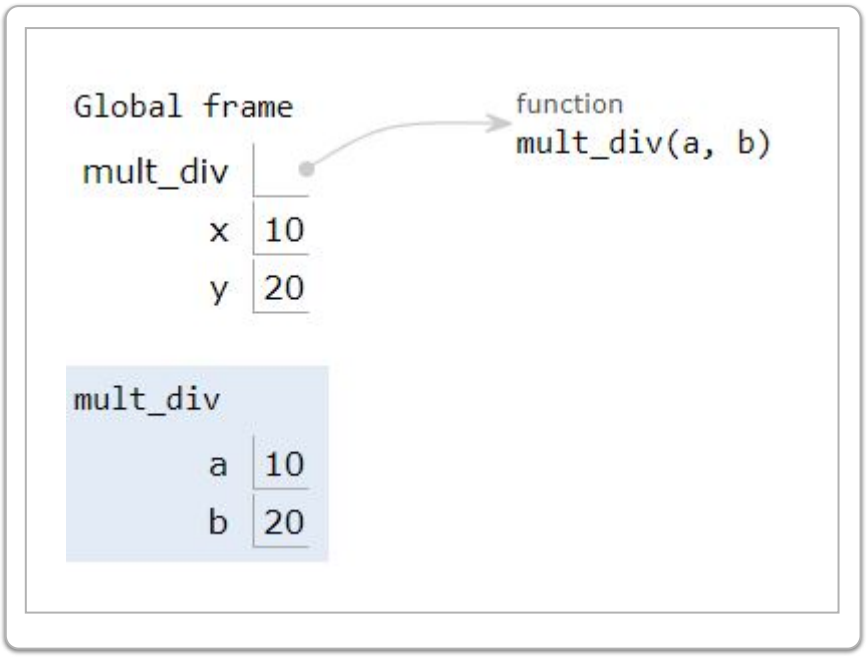


- use *_doc_* method

Parameter Binding

```
def mult_div(a, b):
    """ multiply & divide two numbers """
    result = a * b, a / b
    return result

x = 10; y = 20
result = mult_div(x, y)
```

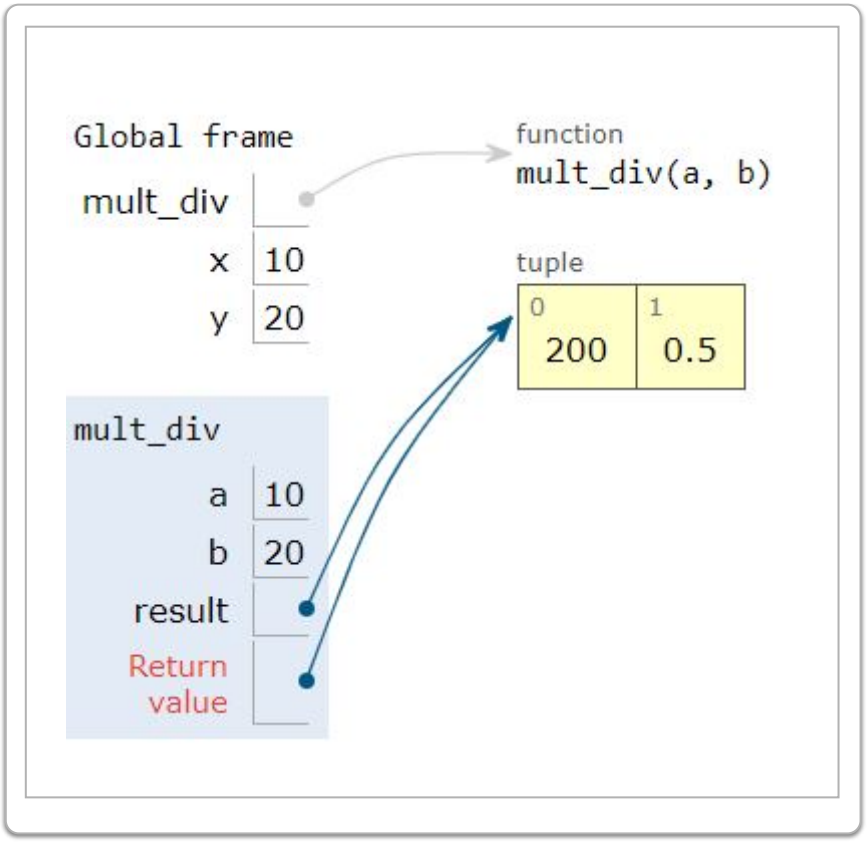


- inputs passed via a tuple

Local Scope

```
def mult_div(a, b):
    """ multiply & divide two numbers """
    result = a * b, a / b
    return result

x = 10; y = 20
result = mult_div(x, y)
```



- *result* inside function

Returning Results

```
def mult_div(a, b):
    """ multiply & divide two numbers """
    result = a * b, a / b
    return result

x = 10; y = 20
result = mult_div(x, y)
```

- results passed via a tuple

Missing *return*

```
def mult_div(a, b):
    """ multiply & divide two numbers """
    result = a * b, a / b
    return result

x = 10; y = 20
result = mult_div(x, y)
```

- result is always *None*

Functions as Objects

```
def mult_div(a, b):
    """ multiply & divide two numbers """
    result = a * b, a / b
    return result

x = 10; y = 20
print(mult_div)
print(mult_div(x, y))
```

- pass functions as arguments

Test Yourself: 5.2.01

An arithmetic progression $A(a, d)$ is a sequence of numbers: $$\begin{align} {x_1} & = {a} \\ {x_2} & = {x_1 + d = a + d} \\ \ldots\ldots\\ {x_n} & = {x_{n-1} + d = a + (n-1)d} \\ \end{align}$$

- Write a function $f(a, d, n)$ to return a list of first n values in $A(a, d)$.
- Generate a list of first 10 values for $a = 5$ and $d = 2$.

Solution:

```
def f(a,d,n):
    """ list of first n elements in arithmetic
        progression with start a and step d """
    last = a + (n -1)* d
```

```
        result = list(range(a, last +1, d))
        return result

    x_list = f(n=10, a=5, d=2)
```

Test Yourself: 5.2.02

A geometric progression \(G(b, q)\) is a sequence of numbers: \begin{align} {y_1} & = {b} \\ {y_2} & = y_1 \cdot q = b \cdot q \\ \ldots\ldots \\ {y_n} & = {y_{n-1} \cdot q = b \cdot q^{n-1}} \\ \end{align}

- Write a function \(g(b, q, n)\) to return a list of first \(n\) values in \(G(b, q)\).
- Generate a list of first 10 values for \(b = 5\) and \(d = 2\).

Solution:

```
def g(b,q,n):
    """ list of first n elements in geometric
    progression with start b and factor d """
    result = [b*q **(i -1) for i in range (1, n +1)]
    return result

y_list = g(n=10, b=5, q=2)
```

Parameter Passing

Parameter Passing

- Parameters are input values passed to functions.
- Several methods are available.
- Passing parameters in Python is different from other languages.

Parameters by Position

```
def mult_div(a, b):
    """ multiply & divide two numbers """
```

```
    result = a * b, a / b
    return result

x = 10; y = 20
result = mult_div(x, y)
```

- parameters bound by position (default)

Parameters by Keyword

```
def mult_div(a, b):
    """ multiply & divide two numbers """
    result = a * b, a / b
    return result

x = 10; y = 20
result = mult_div(b = x, a = y)
```

- parameters bound by keywords

Parameters by Dictionary

```
def mult_div(a, b):
    """ multiply & divide two numbers """
    result = a * b, a / b
    return result

kwargs = {'a': 10, 'b': 20}
x, y = mult_div(**kwargs)
```

- syntax: function(**dict)