CS521 O2
Information Structures with Python

Lecture 9

Guanglan Zhang

guanglan@bu.edu

Some slides adapted from Prof. Eugene Pinsky

Table of Content

Python functions

There are two parts to a Python function: the definition and the invocation

Define a function in Python:

- Use *def* keyword, followed by function name, an optional, comma-separated list of parameters, and : that denotes the end of the function header
- Docstring – a string comment appearing in the first line after the class or method header
- Statement(s) to do some calculation/action
- Optional *return* statement

To invoke a function, we use function name followed by parentheses. A list of arguments may or may not be included inside the parentheses.

Two common paradigms for passing an argument

- A parameter is the variables listed inside the parentheses in the function definition
- An argument is the value that are sent to the function when it is called
- Passing by value
  - ✓ A copy of the argument is passed to the function
  - ✓ Changes made to the parameters inside the function does not affect the original value in the calling environment
- Passing by reference
  - ✓ A reference to the argument is passed to the function
  - ✓ Any changes the function makes to the corresponding parameter will affect the value in the calling environment.
- Python uses pass-by-assignment as parameter names are bound to objects on function entry, and assignment is also the process of binding a name to an object

Parameter passing in Python

- A namespace is a collection of pairs. Each pair consists of a symbolic name and the object associated with the name. This association is called a reference – the name references the object.

- In Python, when passing an argument, it is the reference that get copied, not a new copy of the object itself

- Passing an immutable object, like an int, str, tuple, or frozenset, to a function acts like passing-by-value. The function can't modify the object in the calling environment.

- When passing a mutable object, such as a list, dict, or set, the function can't reassign the object wholesale, but it can change items in place within the object, and these changes will be reflected in the calling environment

- In Python, it's possible to modify an argument from within a function so that the change is reflected in the calling environment. This is referred to as a side effect.

Parameter passing in Python (cont.)

- When a parameter is listed without assignment, it is a required parameter
- When a parameter has a default value assigned, it is an optional parameter
- Several ways do parameter passing in Python
  - ✓ Default parameters - allow some arguments to be omitted when calling the function
  - ✓ By position - must agree in order and number with the parameters declared
  - ✓ By keyword - must agree with declared parameters in number
- Note that default parameter values are only evaluated once when the function is defined. Thus, never use a default value that is mutable.
- Try not to mix positional and keyword arguments
- If you have to mix them, provide all positional arguments first, then keyword arguments

Having a varying number of parameters

- When a parameter name in a Python function definition is preceded by an *, it indicates argument tuple packing.
- *args (placing a single star in front of a parameter name) means any extra positional arguments are packed into a tuple and associated with that parameter
- Note that we cannot use a started parameter as a keyword argument
- The double asterisk (**) can be used with function parameters to specify dictionary packing – use to pass a varying number of keyword arguments to a function
- **kwargs allows use to mix positional and keyword arguments
- *args and **kwargs can be used in the same function, but *args need to proceeds **kwargs

Python special variables

- There are a number of special variables and methods whose name is preceded and followed by __ (two underscores before and after)
- __name__ defines the namespace that a Python module is running in
- When we run the script, the __name__ variable equals __main__
- When we import the containing script, __name__ variable equals the name of the script
- __doc__ prints out the docstring that appears in a class or method

Interrupts, Syntax Errors, and Exceptions

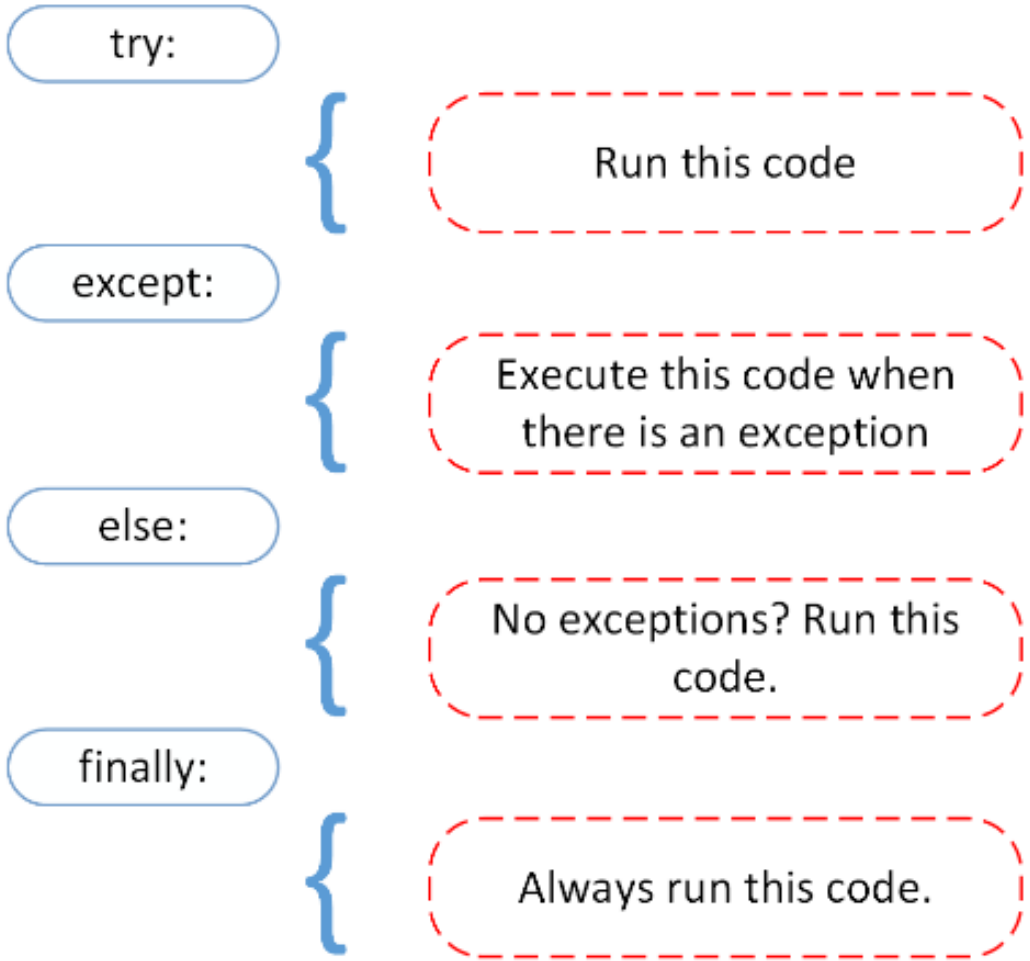
- They all change program flow
- Interrupts are caused by external events ,such as KeyboardInterrupt
- An error are caused by the program itself. It can be a syntax error or an exception
- The Python interpreter finds any invalid syntax during the parsing stage
- If your code is free of Syntax Error, you may get other exceptions raised
- Unhandled exceptions stop program execution

Common Exceptions

Name	Description
Exception	base class
ArithmeticError	errors in computation
FileNotFoundError	no such file or directory
ImportError	import statement fails
IndentationError	improper indentation
IndexError	index not in sequence
KeyError	key not in dictionary
NamedError	identifier not found
SyntaxError	error in syntax
TypeError	unsupported operand type(s)
ZeroDivisionError	division by zero

try and except Block

- The *try* and *except* block is used to catch and handle exceptions
- Using the *else* statement, we can execute a block of code only in the absence of exceptions
- The *finally* clause allows us to execute those clean-up actions that we want to run regardless whether we get exceptions or not



Exercises

1. An arithmetic progression A(a, b) is a sequence of numbers, where
   $x_1 = a$, $x_2 = x_1 + b = a + b$, ..., $x_n = x_{n-1} + b = a + (n - 1)*b$

   Write a function f(a, b, n) to return a list of the first n values in A(a, b, n)
   Generate a list of the first 10 values for a = 5 and b = 3.

2. Handle possible errors that could be raised when using the above function.

3. Write a function ratio_list() to compute the ratio of the 2nd element divided by the 1st element in a list, and return the ratio. The function needs to catch the following errors and print appropriate messages.
   (a) IndexError
   (b) ZeroDivisionError
   (c) Any other errors

If an exception was raised, return None.

Key takeaways

- In Python, when passing an argument, it is the reference that get copied, not a new copy of the object itself
- Passing an immutable object to a function acts like passing-by-value - The function can’t modify the object in the calling environment
- When passing a mutable object, the function can change items in place within the object, and these changes will be reflected in the calling environment
- The try and except block is used to catch and handle exceptions