Functions (*Incomplete*)

Introduction to Computer Programming (Python)

Week 6

Note: using Python 3.11

Vivatsathorn Thitasirivit

Rev. 1.0 (Course 1/2023)

[*https://vt.in.th*](https://vt.in.th)

Outlines
Functions

- Basics of Functions
- Recursion and Recursive Functions
- Lambda Functions
- First-class Functions
- Nested Functions
- Pure and Impure Functions
- Argument's Type Annotation
- Function Arguments: args-kwargs?, default arguments
- Error Handling: try-except-else-finally

Python: Functions

Basics of Functions

Imagine you have a piece of code which appears repeatedly throughout your document.

For example, you have to repeatedly add [1, 2, 3, 4] to and reverse an existing list. You could implement it in a function instead of copying and pasting statements.

The concept is similar to introducing a variable for a constant, so you can change it in one place, applying everywhere.

```
list1 = [9, 7]
list2 = [1, 3, 5]
list3 = [4]
list4 = []
```

```
list1.append([1, 2, 3, 4])
list1.reverse()
list2.append([1, 2, 3, 4])
list2.reverse()
list3.append([1, 2, 3, 4])
list3.reverse()
list4.append([1, 2, 3, 4])
list4.reverse()
```

↓

```
fn(list1)
fn(list2)
fn(list3)
fn(list4)
```

Basics of Functions

Maps

A map in Python and other programming languages behaves and is defined similarly to a map in mathematics.

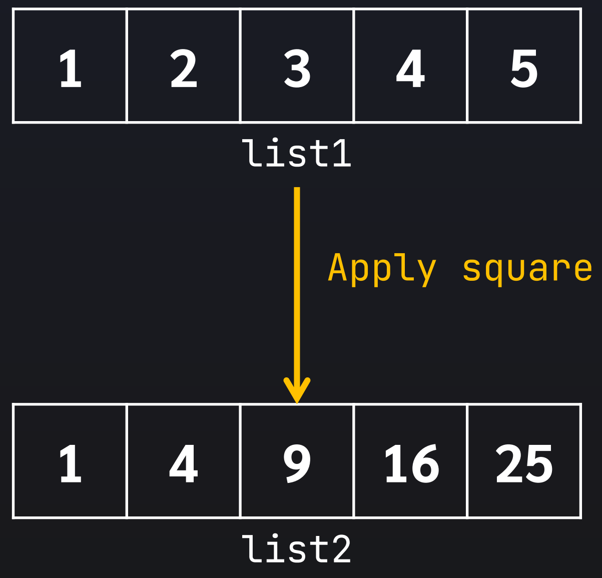
From the last example, we can apply a square function.
(A map is generalization of a function.)

We define a map square defined as follows.

$$\mathrm{square} : \mathbb{Z} \to \mathbb{Z} : \mathrm{square}(x) = x^2$$

The map square takes x and outputs x^2, i.e., a square function.

```python
def square(x):
    return x ** 2
```



Python: Functions

Basics of Functions

Declarations

In Python, you can declare a function using keyword "def," followed by a function name, arguments, and colon.

Body of a function is *indented* from the body. The indentation can be any spaces as long as the whole document uses the same indentation. (Recommended: 2 spaces or 4 spaces.)

- Functions can have arguments.
- Functions can return values.

```
def function1():
    print('Hello world!')


def mul(a, b):
    return a * b


def is_odd(x):
    if x % 2 != 0:
        return True
    else:
        return False
```

Python: Functions

Basics of Functions

Pass Statement

"Pass" in Python is literally "do nothing."

```python
def func():
    pass
```

Recursion

$$F_N = \begin{cases} F_{N-1} + F_{N-2} & ; N > 2 \\ 1 & ; N \leq 2 \end{cases}$$

A recursion is a process of defining a problem in terms of itself (typically simpler version of itself).

It consists of

1. Base Case
2. Recursive Case

For example, a Fibonacci N^{th} number in a sequence can be calculated using this recursive definition:

$$F_N = F_{N-1} + F_{N-2}$$

where $N > 1$ and $F_1 = F_2 = 1$.

Python: Functions
Recursion

$$F_N = \begin{cases} F_{N-1} + F_{N-2} & ; N > 2 \\ 1 & ; N \leq 2 \end{cases}$$

Recursive Functions

$$\text{factorial}(n) = \begin{cases} 1 & ; n = 0 \\ n \cdot \text{factorial}(n-1) & ; n > 1 \end{cases}$$

Example, a factorial function is typically defined as:

$$n! = n(n-1)(n-2)\ldots 1$$

where $0! = 1$.

But you can see that the part following n is just $(n-1)!$.

$$n! = n(n-1)!$$

which is also a recursion. If we define it as a function:

$$\text{factorial}(n) = n \cdot \text{factorial}(n-1)$$

then it is a "Recursive Function."

Recursive Functions

$$\text{factorial}(n) = \begin{cases} 1 & ; n = 0 \\ n \cdot \text{factorial}(n-1) & ; n > 1 \end{cases}$$

Recursive Functions

Implementing recursive factorial function in Python:

```python
def factorial(n):
    if n == 0:
        # Base Case
        return 1
    else:
        # Recursive Case
        return n * factorial(n - 1)
```

$$\text{factorial}(n) = \begin{cases} 1 & ; n = 0 \\ n \cdot \text{factorial}(n-1) & ; n > 1 \end{cases}$$