CS1302 - Lecture 5 Functions
(Dr Helena WONG)
==========

[Summary of Topic 5] Functions

[Helena's supplements for introduction]

[Functions.ipynb]

1. **What is a Function?**
2. **Code Reuse**
   - Perfect Square
   - Integer Square Root
3. **Modules**
   - Builtins Module
   - Importing Eternal Modules
4. **Documentation**

==========

[Helena's supplements for introduction]

function call and (): The input() example

```
========================
x = input("give me a number")
print(x,x,x)
========================
x = input()
print(x,x,x)
========================
x = input
print(x,x,x)
========================

print('1st input:',input(),'2nd input:',input()) # 4 arguments
# x1 = input()
# x2 = input()
# print('1st input:', x1, '2nd input:', x2)
```

==========

The lecture notebook [Functions.ipynb]:

1. What is a Function?

callable – A function is a *callable* object

```
callable(callable), callable(1) # True, False
```

==========

def

```
def sayItTwice(x):
    print(x)
    print(x)

sayItTwice("hello")
sayItTwice(123)
```

```
====================
def increment(x):
    return x + 1

a = increment(3)
b = sayItTwice("hello")

print(a, b) # 4 None

====================
def length_of_hypotenuse(a, b):
    return (a ** 2 + b ** 2) ** 0.5

a = length_of_hypotenuse(1, 2)
b = length_of_hypotenuse(3, 4)
```

==========

return vs change

```
def increment(x):
    return x + 1

def increment2(x):
    x += 1      # i.e. x = x+1

x = 3
a = increment(x)
b = increment2(x)
print(a, b, x)    # output is _____
```

==========

Python functions are **First-class citizens**

1. **assigned to a variable,**
2. **passed as an input argument, and**
3. **returned by a function.**

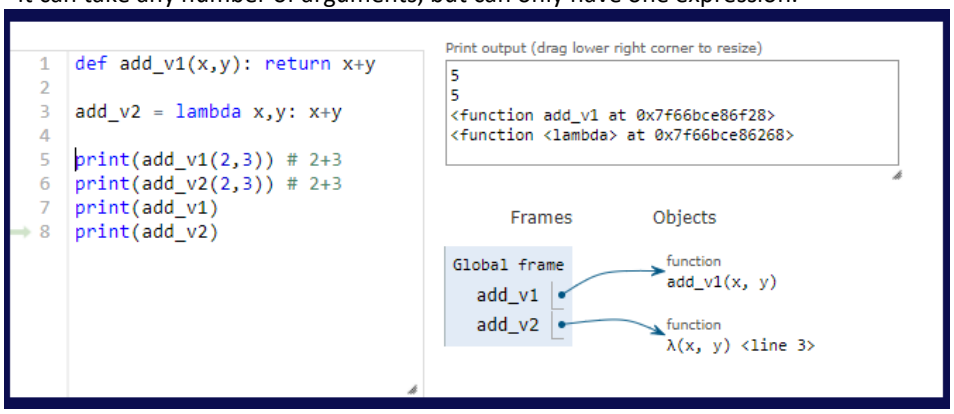
An identity function

```
def i(x):
    return x

j1 = i
j2 = i(i)

print(j1 == i)    # True
print(j2 == i)    # True

print(i.__name__)
attr = dir(i) # dir: the attributes etc. of an object
```

==========

A function as a **Lambda expression**

A lambda expression creates a small anonymous function.
It can take any number of arguments, but can only have one expression.




```
def add_v1(x, y): return x + y

add_v2 = lambda x, y: x + y
print(add_v1(2, 3))    # 2+3
print(add_v2(2, 3))    # 2+3
print(add_v1)
print(add_v2)
```

==========

Infinite loop with lambda? [Courseware => Jupyter book for CS1302]

Why the name <lambda>? [Courseware => Jupyter book for CS1302]

==========

==========

2. Code Reuse

==========

Perfect Square

```
for i in range(10):
    print(i**2)
```

==========

is_perfect_square(n)

```
# slow
def is_perfect_square(n):
    for i in range(n):
        if i**2 == n:
            return True
    return False

x1 = is_perfect_square(10**2)
x2 = is_perfect_square(10**2 + 1)
# x3 = is_perfect_square(10**10)
# x4 = is_perfect_square(10**10 + 1)
# x5 = is_perfect_square(10**100)
# x6 = is_perfect_square(10**100 + 1)
```

==========

the package `wrapt_timeout_decorator`

```
==========================
%pip install wrapt_timeout_decorator >/dev/null 2>&1
from wrapt_timeout_decorator import timeout

==========================
duration = 5

@timeout(duration)    # raise error if not complete in 5 seconds.
def test():
    x1 = is_perfect_square(10**2)
    print(x1)
    x2 = is_perfect_square(10**2 + 1)
    print(x2)
    x3 = is_perfect_square(10**10)
    print(x3)
    x4 = is_perfect_square(10**10 + 1)
    print(x4)

test()    # run the test
```

==========

Integer Square Root

An integer n is a perfect square iff n is the square of its integer square root.

```
import math

# Quick but __ !
def is_perfect_square(n):
    # check if n is the square of its integer square root
    # return n == int(n**0.5) ** 2
    return math.isclose(n, int(n**0.5) ** 2)

x1 = is_perfect_square(10**10)
x2 = is_perfect_square(10**10+1) # should be False
print(x1,x2)

# not is_perfect_square(10**10+1) fails because tolerance too high.
x = 10**100
int((x) ** 0.5)
```

Solution: isqrt

```
x = 10**100
math.isqrt(x), int((x) ** 0.5)
```

An example of Code-Reuse!!

The math module consists mostly of thin wrappers around the platform C math library functions. - pydoc last paragraph

[Ref: Wikipedia] CPython

The default and most widely used implementation of the Python language is CPython: CPython is the reference implementation of the Python programming language, using C functions.

==========

3. Modules

Function vs Module vs Package/Library

```
import math
```

```
math.isclose(1.0, 1.0000001)

import matplotlib.pyplot
matplotlib.pyplot.stem([4,3,2,1])
matplotlib.pyplot.ylabel(r'$x_n$')
matplotlib.pyplot.xlabel(r'$n$')
matplotlib.pyplot.title('A sequence of numbers')
matplotlib.pyplot.show()
```

To facilitate code reuse, all Python codes are organized into libraries called *modules*. E.g., you can list all available modules

To list all modules (not including built-in modules like math):

```
%pip list

%pip show matplotlib

# The search path of Python's packages
import sys
sys.path

# For additional packages
%pip install …

# ASCII art of a cow

    %pip install cowsay

    import cowsay
    cowsay.cow("I am a pip installed package ((((((...ip)ip)ip)ip)ip)ip)!")
```

==========

Builtins Module and Importing External Modules

==========

How to import functions from a library?

try in OPTMentor

```
    print(pow(3,2))

    import math
    print(math.pow(3,2)) # float!

    from math import pow
    print(pow(3,2))

    from math import * # from math import pow, sin, pi
    print(sin(pi))
```

try in a notebook:

```
    ?pow

    ?__builtin__
```

```
        # Indeed, every function must come from a module.
        # built-in functions are from _builtin_ module

        import math
        ?math.pow

        from math import pow
        ?pow

        #    * using dir:
        print(dir(__builtin__)) # attributes and functions of __builtin__
        print(dir()) # the names in the current scope

* handling complex number:
        x1 = __builtin__.pow(-3, .5)    # ok
        x2 = math.pow(-3, .5)    # error

* from math import pow:
* from math import *:
        ==> __builtin__'s pow is polluted due to name collision

        ========================
* use as:
        from math import pow as fpow
        print(pow(3,2))
        print(fpow(3,2))

        import math as m
        print(pow(3,2))
        print(m.pow(3,2))

        ========================
        import package.module_name as short_name
        from package_name import module_name as short_name

        import matplotlib.pyplot as plt
        plt.stem([4,3,2,1])
        plt.ylabel(r'$x_n$')
        plt.xlabel(r'$n$')
        plt.title('A sequence of numbers')
        plt.show()
        # from matplotlib import pyplot as plt #equivalent to above

        ========================
        # Caution:
        import math as m

        for m in range(5):
            print(m.pow(m, 2))

    ========================
```

4. Documentation

```
========================
documentation:
```

```
        def greet():
            ''' just
            to greet
            somebody'''
            print("hi!")

        help(greet)

        # For more, read Writing Functions.ipynb, Python style guide (PEP 257)
========================
```

Annotate the function with

hints of the types of the arguments and return value

```
        def increment(x: int) -> int:
            return x + 1    # + operation is used and may fail for 'str'

        my_mark = 3 # 3?    3.5? "three"?
        new_mark = increment(my_mark)
        my_mark, new_mark

========================
```