

# D002 Python for Everyone

## Lesson 4: Function

Dr. Kevin Wang

Department of Computer Science and Engineering

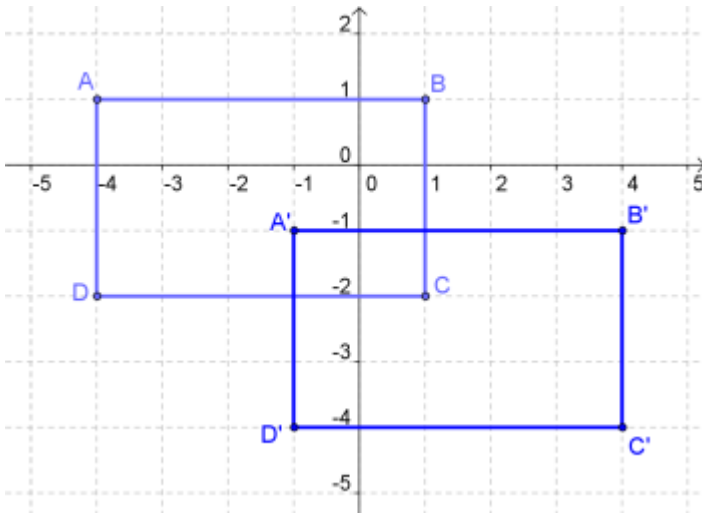
# What we have covered so far

- Variables
- Basic Operation `= + - * / ** %`
- Comparison symbol `== != > < >= <=`
- Logical Operator `and or not`
- Branching `if if else if elif elifi elifi else`
- Loop `while for`
- List
  - Index always starts from 0!

## Warm up exercise

Open [Q1.py](#) and write one line of code for each of the subquestions.

# Do you know coordinate system?



Let's define the position and size of a Window using its four corners, or may be just two - A and C.

Question: Can we know if a Window is contained within another Window?

## Not Containing, just overlapping

```
# Rectangle 1
a1 = [1, 8]    # b1 = [4, 8] ; d1 = [1, 3]
c1 = [4, 3]
# Rectangle 2
a2 = [3, 11]   # b2 = [6, 11] ; d2 = [3, 4]
c2 = [6, 4]
```

## Containing

```
# Rectangle 1
a1 = [1, 18]   # b1 = [8, 18] ; d1 = [1, 3]
c1 = [8, 3]
# Rectangle 2
a2 = [3, 11]   # b2 = [6, 11] ; d2 = [3, 4]
c2 = [6, 4]
```

# Definition of Containing

If all corner points of a rectangle `x` are *inside* another rectangle `y`, we say `y` contains `x`

```
#Assume a and c are two points of Y; t is a point
if a[0] < t[0] < c[0] and a[1] > t[1] > c[1]:
    print("t is inside the rectangle formed by a,c")
```

=>

In fact checking only the two diagonal points are sufficient (hmm, slightly better)

# It is so messy

```
# Assume we have a1 c1 a2 c2 defined elsewhere

if a2[0] <= a1[0] <= c2[0] and a2[0] <= c1[0] <= c2[0] and \
    a2[1] >= a1[1] >= c2[1] and a2[1] >= c1[1] >= c2[1]:
    print("Rectangle 2 contains Rectangle 1")
elif a1[0] <= a2[0] <= c1[0] and a1[0] <= c2[0] <= c1[0] and \
    a1[1] >= a2[1] >= c1[1] and a1[1] >= c2[1] >= c1[1]:
    print("Rectangle 1 contains Rectangle 2")
```

Codes are repeating, why would we code it again and again?

**DRY**: Don't Repeat Yourself

# Function - to Modularize

- A function is a piece of code in a program. It performs a specific task.
- Good for 'Do-not-repeat yourself' (DRY) and 'divide-n-conquer' (modular design) of the program
- There are two basic types of functions. Built-in functions and user-defined ones. The built-in functions are part of the Python language.

Example of built-in function that you already know: `ceil` , `sqrt` , `range` , `randint`



# Building our own functions

- We create a new function using the `def` keyword followed by optional parameters in parenthesis.
- This defines the function but does not execute the body of the function
- Definition vs. usage: once we have defined a function, we can call (or invoke) it as many times as we like

```
def average(a, b): # function name: average; parameters: a, b  
    return (a + b) / 2 # return value
```

```
print(average(3, 10))  
x = average(5, 10)  
y = average(x, 10)  
z = average(x, average(x, y))  
print(x, y, z)
```

## Another example - Fortunate Teller

```
def luck_level(day, month):  
    if day + month % 5 > 3:  
        print("You have good luck today")  
    else:  
        print("You should sleep more")  
  
luck_level(15, 3)  
luck_level(4, 1)
```

This function does not return any value, we call it a `void` function

## Another example - Inside

```
def is_inside(a, c, t):  
    if a[0] <= t[0] <= c[0] and a[1] >= t[1] >= c[1]:  
        return True      # when it returns, the function stops  
    return False  
  
# Rectangle 1  
a1 = [1, 18]    # b1 = [8, 18] ; d1 = [1, 3]  
c1 = [8, 3]  
t = [4, 4]  
if is_inside(a1, c1, t):  
    print("t is inside rectangle 1")  
else:  
    print("t is outside rectangle 1")
```

```
def is_inside(a, c, t):  
    if a[0] <= t[0] <= c[0] and a[1] >= t[1] >= c[1]:  
        return True      # when it returns, the function stops  
    return False  
  
# Rectangle 1  
a1 = [1, 8]    # b1 = [4, 8] ; d1 = [1, 3]  
c1 = [4, 3]  
# Rectangle 2  
a2 = [3, 11]   # b2 = [6, 11] ; d2 = [3, 4]  
c2 = [6, 4]  
  
if is_inside(a1, c1, a2) and is_inside(a1, c1, c2):  
    print("Rectangle 2 is inside Rectangle 1")  
elif is_inside(a2, c2, a1) and is_inside(a2, c2, c1):  
    print("Rectangle 1 is inside Rectangle 2")
```

# Return Values

- Often a function will take its arguments, do some computation and return a value to be used as the value of the function call in the calling expression.
- The `return` keyword is used for this.
- The return statement ends the function execution and “sends back” the result of the function
- A **void** function does not return a value, but it can also use return statement

```
def void_function(x):  
    if x == 1:  
        print(x)  
        return  
    while x >= 1  
        x = x / 2  
        print(x)
```

# Parameters

- A function can have 0 or more parameters. It is less likely to have no parameter, but indeed it is possible.

```
def option(): # you need () even there is no parameter
    print("Option 1: Pass\nOption 2: Shoot\nOption 3: Dribble")
    x = input("Please choose between option 1 to 3")
    while x < 1 or x > 3:
        x = input("Wrong input, type again")
    return x
```

## Q2

Write a void function called `factors` that find all factors of an integer `n`

```
# Your code here  
for x in range(1, n + 1):  
    if n % x == 0:  
        print("%d divides %d" % (x, n))  
  
# Example of calling  
factors(40)  
factors(5)
```

## Q3

Write a function called `factors` that find all factors of an integer `n`  
It returns all factors as a list

```
_____ # Your code here
result = []
for x in range(1, n + 1):
    if n % x == 0:
        _____
    _____

# Example of calling
print(factors(40))
a = factors(30)
if 5 in a:
    print("5 is a factor of 30")
```



## Q4

Write a function called `checker` that reveals the index(s) of the sentence of which a letter is located. It returns an empty list if the letter is not inside the sentence.

```
def checker(sentence, letter):  
    result = []  
    # Your code here  
  
    return result  
  
a = checker("Apple", "p") # a = [1, 2]  
b = checker("Banana", "p") # b = []  
c = checker("Cat", "a") # c = [1]
```

## Q5

Write a void function called `printer` that takes two parameters: `secret` and `opened`. `secret` is a string and `opened` is a list of indexes. The function prints all letter of the string `secret` by masking all indexes that is *not* in `opened` as `_`.

```
def printer(secret, opened):  
    i = 0  
    while i < len(secret):  
        # Your code here
```

```
        i = i + 1  
        print()
```

*# Note: You might use `print(x , end="")` to print x without changing line*

```
printer("apple", [1, 2]) # _pp_  
printer("orange", [0, 5]) # o____e  
printer("cat", []) # ____
```