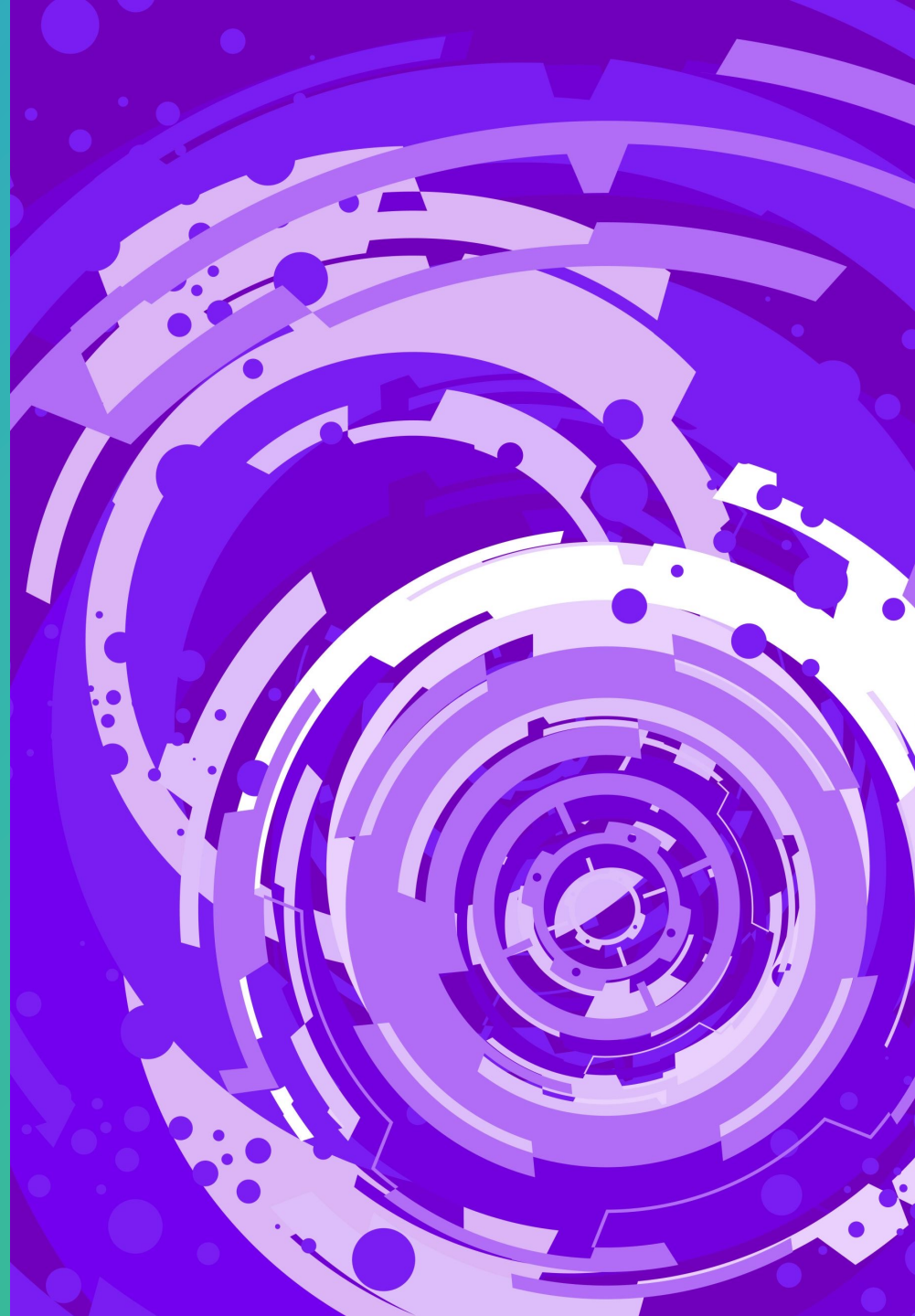
URBAN GEOGRAPHIC INFORMATION SYSTEM

Python Basic II – Conditions & Loops & Functions

Chun-Hsiang Chan

Department of Geography,

National Taiwan Normal University



Outline

- Conditions
- For Loops
- While Loops
- Functions
- Recursive Functions

Conditions

- Usually, we need to use "conditions" to avoid occurring some cases or situations. Sometimes, we just want to classify all items into several categories by following some rules.

```
# we can simply use logical conditions to solve this
a, b = [3, 5]
a == b # equal
a != b # not equal
a < b # less than
a <= b # less than or equal to
a > b # greater than
a >= b # greater than or equal to
```

Conditions

- Python relies on indentation (whitespace at the beginning of a line) to define scope in the code.
- Other programming languages often use curly-brackets for this purpose.

```
# simple condition with if
a, b = [3, 5]
if a == b:
    print("a is equal to b")
```

The whitespace here stands for indentation.
I usually use “tab” button for indentation because it is much simpler and makes consistent to other indentations.

https://www.w3schools.com/python/python_conditions.asp

Conditions

- In most cases, only one “if” cannot satisfy our real-world problems; therefore, here, I introduce other items – “elif and else”.

```
# simple condition with if, elif, else
a, b = [3, 5]
if a == b:
    print("a is equal to b")
elif a > b:
    print("a is larger than b")
else:
    print("a is smaller than b")
```

Conditions

- Are you satisfied with the functionality of "if…else"? I do not think so because some cases require two or more conditions in a single procedure. For example, how to extract all postmenopausal women with single-line code?

```
# two or more conditions
a, b, c, d = [3, 5, 51, 500]
if a == b and a < d:
    print("situ 1")
elif c < b or c > d:
    print("situ 2")
elif not c < b:
    print("situ 3")
```

Conditions

- In addition to multiple condition, we can leverage nested conditions for complicated problems or situations.

```
# nested conditions
a, b, c, d = [3, 5, 51, 500]
if a == b:
    if a < d:
        print("situ 1")
elif c < b or c > d:
    print("situ 2")
elif c > b:
    pass # do nothing
```

Lab Practice 1 (conditions)

- Design a code for determining your GPA of each subject.
- Please try these cases:
  1) 92
  2) 60
  3) 2
  4) 0
  5) 102
  6) -5

  # notice: your code needs to avoid incorrect inputs

Letter	Range	Grade Point
A+	90-100	4.3
A	85-89	4
A-	80-84	3.7
B+	77-79	3.3
B	73-76	3
B-	70-72	2.7
C+	67-69	2.3
C	63-66	2
C-	60-62	1.7
D	50-59	1
E	1-49	0
X	0	0

For Loops

for i in range(end): # start from 0

- In Python, we have two loop functions for item-wise iteration.
- For example, we want to print all numbers ranging from 0 to 100, individually.

```
# for loop
for i in range(100):
    print(i) # does it iterate to 100? If not, how can you fix it

# for loop with a condition
for i in range(100):
    if i/10==0:
        print(i)
```

For Loops

- In some cases, we can directly iterate with other approaches.

```
# for loop with a list
scorels = [78, 80, 100, 89, 50, 65, 70]
for i in scorels:
    print(i) # what does it iterate and output?

# for loop with a string
for i in "taiwan":
    print(i) # what does it iterate and output?
```

For Loops

- If you have some conditions, and then you need other tools.

```
# for loop with conditions and rules
scorels = [78, 80, 100, 89, 50, 65, 70]
for i in scorels:
    if i < 60:
        print(i, "you are failed in this subject!")
    elif i > 100 or i < 0:
        break
    else:
        pass
# change the scores and observe the functionality of break and pass
```

For Loops

- One loop cannot satisfy our requests, ….
- Therefore, we introduce another approach – nested loop.

```
# nested for loop
for i in range(10):
    # print(i)
    for j in range(10):
        print(i, j)
```

```
# nested for loop
(1) i = 0 then j = 0 to 9, respectively
(2) i = 1 then j = 0 to 9, respectively
(3) i = 2 then j = 0 to 9, respectively
(4) i = 3 then j = 0 to 9, respectively
(5) …
(6) …
(7) …
(8) …
```

For Loops

- How can we change the iteration way?

```
for i in range(start, end, hopping_step):
```

```
# hopping with 5 step
for i in range(0, 100, 5):
    print(i)

# reverse hopping
for i in range(100, 0, -10):
    print(i)

# observe the regularity
```

While Loops

- While loops are very different from for loop because of its nature. For example, for loop has a variable that could change in each iteration; however, while loop does not require to do so.
- Without using a changeable variable, how does while loop work?
- And what is the benefit of while loop comparing to for loop?
- Think about this.

While Loops

while ending_condition:

- At the beginning, we demonstrate a simple example…

```
# while loop
i = 0
while i<10:
    print(i)
    i += 1 # equals to i = i + 1
```

```
# infinite while loop
i = 0
while 1:
    if i > 10:
        break # stop iteration
    else:
        print(i)
        i += 1
        continue # keep iteration
```

While Loops

- How about nested while loop?

```
# nested while loop
i = 0
while i < 5:
    j = 0
    while j < 5:
        print(i, j)
        j += 1
    i += 1 # equals to i = i + 1
```

Lab Practice 2 (for and while)

- Make a 9 x 9 multiplication table.

```
1   1 * 1 = 1   2 * 1 = 2   3 * 1 = 3   4 * 1 = 4   5 * 1 = 5   6 * 1 = 6   7 * 1 = 7   8 * 1 = 8   9 * 1 = 9
2   1 * 2 = 2   2 * 2 = 4   3 * 2 = 6   4 * 2 = 8   5 * 2 = 10  6 * 2 = 12  7 * 2 = 14  8 * 2 = 16  9 * 2 = 18
3   1 * 3 = 3   2 * 3 = 6   3 * 3 = 9   4 * 3 = 12  5 * 3 = 15  6 * 3 = 18  7 * 3 = 21  8 * 3 = 24  9 * 3 = 27
4   1 * 4 = 4   2 * 4 = 8   3 * 4 = 12  4 * 4 = 16  5 * 4 = 20  6 * 4 = 24  7 * 4 = 28  8 * 4 = 32  9 * 4 = 36
5   1 * 5 = 5   2 * 5 = 10  3 * 5 = 15  4 * 5 = 20  5 * 5 = 25  6 * 5 = 30  7 * 5 = 35  8 * 5 = 40  9 * 5 = 45
6   1 * 6 = 6   2 * 6 = 12  3 * 6 = 18  4 * 6 = 24  5 * 6 = 30  6 * 6 = 36  7 * 6 = 42  8 * 6 = 48  9 * 6 = 54
7   1 * 7 = 7   2 * 7 = 14  3 * 7 = 21  4 * 7 = 28  5 * 7 = 35  6 * 7 = 42  7 * 7 = 49  8 * 7 = 56  9 * 7 = 63
8   1 * 8 = 8   2 * 8 = 16  3 * 8 = 24  4 * 8 = 32  5 * 8 = 40  6 * 8 = 48  7 * 8 = 56  8 * 8 = 64  9 * 8 = 72
9   1 * 9 = 9   2 * 9 = 18  3 * 9 = 27  4 * 9 = 36  5 * 9 = 45  6 * 9 = 54  7 * 9 = 63  8 * 9 = 72  9 * 9 = 81
```

- Another style.

```
11  1 * 1 = 1   1 * 2 = 2   1 * 3 = 3   1 * 4 = 4   1 * 5 = 5   1 * 6 = 6   1 * 7 = 7   1 * 8 = 8   1 * 9 = 9
12  2 * 1 = 2   2 * 2 = 4   2 * 3 = 6   2 * 4 = 8   2 * 5 = 10  2 * 6 = 12  2 * 7 = 14  2 * 8 = 16  2 * 9 = 18
13  3 * 1 = 3   3 * 2 = 6   3 * 3 = 9   3 * 4 = 12  3 * 5 = 15  3 * 6 = 18  3 * 7 = 21  3 * 8 = 24  3 * 9 = 27
14  4 * 1 = 4   4 * 2 = 8   4 * 3 = 12  4 * 4 = 16  4 * 5 = 20  4 * 6 = 24  4 * 7 = 28  4 * 8 = 32  4 * 9 = 36
15  5 * 1 = 5   5 * 2 = 10  5 * 3 = 15  5 * 4 = 20  5 * 5 = 25  5 * 6 = 30  5 * 7 = 35  5 * 8 = 40  5 * 9 = 45
16  6 * 1 = 6   6 * 2 = 12  6 * 3 = 18  6 * 4 = 24  6 * 5 = 30  6 * 6 = 36  6 * 7 = 42  6 * 8 = 48  6 * 9 = 54
17  7 * 1 = 7   7 * 2 = 14  7 * 3 = 21  7 * 4 = 28  7 * 5 = 35  7 * 6 = 42  7 * 7 = 49  7 * 8 = 56  7 * 9 = 63
18  8 * 1 = 8   8 * 2 = 16  8 * 3 = 24  8 * 4 = 32  8 * 5 = 40  8 * 6 = 48  8 * 7 = 56  8 * 8 = 64  8 * 9 = 72
19  9 * 1 = 9   9 * 2 = 18  9 * 3 = 27  9 * 4 = 36  9 * 5 = 45  9 * 6 = 54  9 * 7 = 63  9 * 8 = 72  9 * 9 = 81
```

Functions

def function_name(input_arg):

- Sometimes, you have to do something many times; however, there is no built-in package or function that helps you.
- As a result, you need to design the customized function by yourself.

```
# typical function
def cm2m(cm):
    m = cm/100
    return m

# use typical function
cm2m(120)
```

```
# simple function
def cm2m_(cm):
    return cm/100

# use simple function
cm2m_(120)
```

Functions

- Here, we introduce "local variable" and "global variable".
- All variables in the function indentation are local variables which indicates that they cannot be used outside the block.
- Meanwhile, the all variables used outside the function cannot be used in the function.

global

local

global

```
# observe the variables
cm = 1000
m = 900
a = 25
print(cm, m, a)

def cm2m(cm):
    global a, x
    a, x = 49, 13
    m = cm/100
    print(cm, m, a, x)
    return m

cm2m(120)
print(cm, m, a, x)
```

Functions

- Why do we need a local variable?
- Why do we need a global variable?
- Please give the reason with examples.

Recursive Functions

- Recursive function is a powerful approach to get some results with special rules or regularities.

```
# recursive function
def my_sum(a):
    if a == 1 or a == 0:
        return a
    else:
        return a + my_sum(a-1)

# use recursive function
my_sum(10)
```

```
# if a = 4, then …
4 + my_sum(4-1) # 4 - 1 = 3
4 + (3 + my_sum(3-1))
4 + (3 + (2 + my_sum(2-1)))
# which is 1
# Because … my_sum(2-1) = 1
# So …
4 + (3 + (2 + 1)
4 + (3 + 3)
4 + 6
10
```

Lab Practice 3 (recursive function)

- Design a function that can calculate the **factorial** answer.
- Example: **my_factorial(5) = 120**

```
# recursive function
def my_factorial(a):
    ...
    ...
    ...
    ...
```

Lab Practice 4 (recursive function)

- Design a function that generates a **fibonacci** number.
- **my_fibon(10)** # 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …

```
# recursive function
def my_fibon(a):
    ...
    ...
    ...
    ...
```

Question Time

- **Assignment:**
- **Download today's lab practice and upload to moodle.**
- **Thx**

The End

Thank you for your attention!

Email: chchan@ntnu.edu.tw
Web: toodou.github.io

SCAN ME