ITD62-123 COMPUTER PROGRAMMING

IMI62-122 FUNDAMENTAL OF COMPUTER PROGRAMMING

Theerat Saichoo & Yunyong Punsawad
School of Informatics, Walailak University

Chapter 3
Loops

python™

Topics

- Collections of data
  - Python Lists
  - Python Tuples
  - Python Sets
- for loops
- while loops

PART 1: Collections of data

Collections of data

- There are four collection data types in the Python programming language:
  - **List** is a collection which is ordered and changeable. Allows duplicate members.
  - **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
  - **Set** is a collection which is unordered and unindexed. No duplicate members.
  - **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

Python lists

- Lists are used to store multiple items in a single variable.
- Lists are created using square brackets:

```
mylist = ["apple", "banana", "cherry"]
```

- List items are ordered, changeable, and allow duplicate values.
- List items are indexed, the first item has index [0], the second item has index [1] etc.
- List can contain different data types.

Python lists

- Examples
- list1 = ["apple", "banana", "cherry"]
- list2 = [1, 5, 7, 9, 3]
- list3 = [True, False, False]
- list4 = ["abc", 34, True, 40, "male"]

Python lists

- **Access Items** - List items are indexed and you can access them by referring to the index number:

```
thislist = ["apple", "banana", "cherry"]
```

- **Note 1:** The first item has index 0.

```
print(thislist[1])
```

- **Result:** banana
- **Note 2:** Negative indexing means start from the end

```
print(thislist[-1])
```

- **Result:** cherry

Python lists

- **Range of Indexes -** We can specify a range of indexes by specifying where to start and where to end the range.

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

- Result: ['cherry', 'orange', 'kiwi']

- *Note: Remember that the first item is position 0, and note that the item in position 5 is NOT included*

Python lists

- **Range of Indexes -** By leaving out the start value, the range will start at the first item:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:4])
```

- Result: ['apple', 'banana', 'cherry', 'orange']

- *Note: This will return the items from index 0 to index 4.*

Python lists

- **Range of Indexes -** By leaving out the end value, the range will go on to the end of the list:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:])
```

- Result: ['cherry', 'orange', 'kiwi', 'melon', 'mango']

- *Note: This will return the items from index 2 to the end.*

Python lists

- **Change Item Value** - To change the value of a specific item, refer to the index number:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

- **Result:** ['apple', 'blackcurrant', 'cherry']

Python lists

- **Append Items -** To add an item to the end of the list, use the append() method:

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

- **Result:** ['apple', 'banana', 'cherry', 'orange']

Python lists

- **Remove List Items**

```
thislist = ["apple", "banana", "cherry"]
```

- **Remove Specified Item**

```
thislist.remove("banana")
```

- **Remove Specified Index**

```
thislist.pop(1)
```

- **Result:** ['apple', 'cherry']

Python lists

- **Clear the List**
- The clear() method empties the list.

```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
```

- **Result:** Empty list

Python Tuples

- A tuple is a collection which is **ordered** and **unchangeable**.
- In Python tuples are written with round brackets.

```
mytuple = ("apple", "banana", "cherry")
```

- Access Items – same as list

Python Sets

- A set is a collection which is **unordered** and **unindexed**.
- In Python, sets are written with curly brackets.

```
myset = {"apple", "banana", "cherry"}
```

- Access Items – no indexed, no key, using loop

Special thanks to W3Schools

Class activity 4

เขียนคำสั่งภาษา Python เพื่อให้มีการทำงานของโปรแกรมดังนี้ ...

1. สร้าง List ซึ่งประกอบไปด้วยข้อมูล 5 ตัว คือ Jessy, Ammy, Luzy, Jenny, John
2. เพื่อเรียกใช้ข้อมูล Ammy โดยการอ้างอิง Index
3. เพื่อลบข้อมูล Jenny ออกจาก list โดยการอ้างอิง Index
4. เพื่อเพิ่มข้อมูล BigBoss ต่อท้าย list
5. เพื่อแสดงสมาชิกเฉพาะ Luzy, Jenny และ John
6. เพื่อแสดงสมาชิก Jessy และ Ammy

PART 2: Loops

while loops

- **The while loops** - With the while loop we can execute a set of statements as long as a condition is true.
- **Syntax**

```
while condition:
        command statements
```

- **Example**

```
i = 1
while i < 6:
        print(i)
        i += 1
```

**** The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.*

while loops

- **The break Statement -** With the break statement we can stop the loop even if the while condition is true:
- **Example**

```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```

- Exit the loop when i is 3:

while loops

- **The continue Statement** - With the continue statement we can stop the current iteration, and continue with the next:
- **Example**

```
i = 0
while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)
```

- Continue to the next iteration if i is 3:

while loops

- **The else Statement** - With the else statement we can run a block of code once when the condition no longer is true:
- **Example**

```
i = 1
while i < 6:
        print(i)
        i += 1
else:
        print("i is no longer less than 6")
```

- Print a message once the condition is false:

for loops

- **Python for Loops**
- A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
- This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.
- With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

for loops

- Syntax

```
for … in … :
        command statements
```

- Example

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
        print(x)
```

```
apple
banana
cherry
```

- Print each fruit in a fruit list:
- The for loop does not require an indexing variable to set beforehand.

for loops

- **Looping Through a String -** Even strings are iterable objects, they contain a sequence of characters:
- **Example**

```
for x in "banana":
    print(x)
```

- Loop through the letters in the word "banana":

```
b
a
n
a
n
a
```

for loops

- **The break Statement** - With the break statement we can stop the loop before it has looped through all the items:
- **Example**

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
    if x == "banana":
        break
```

```
apple
banana
```

- Exit the loop when x is "banana":

for loops

- **The continue Statement** - With the continue statement we can stop the current iteration of the loop, and continue with the next:
- Example

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    if x == "banana":
    continue
print(x)
```

```
apple
cherry
```

- Do not print banana:

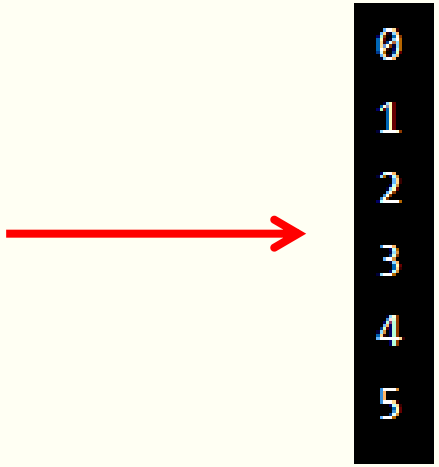
for loops

- **The range() Function**
- To loop through a set of code a specified number of times, we can use the range() function,
- The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.
- It is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):

for loops

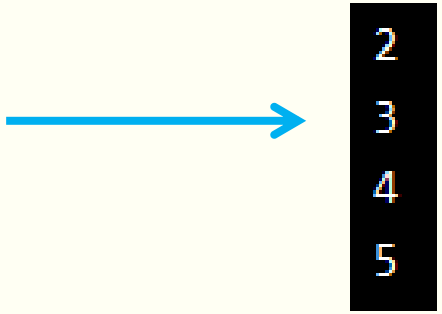
- Example #1

```
for x in range(6):
    print(x)
```



- Example #2

```
for x in range(2, 6):
    print(x)
```



for loops

- The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, **3**):
- Example

```
for x in range(2, 30, 3):
    print(x)
```

End

increment

Start

```
2
5
8
11
14
17
20
23
26
29
```

for loops

- **Else in For Loop** - The else keyword in a for loop specifies a block of code to be executed when the loop is finished:
- Example

```
for x in range(6):
    print(x)
else:
    print("Finally finished!")
```

```
0
1
2
3
4
5
Finally finished!
```

- Print all numbers from 0 to 5, and print a message when the loop has ended:

for loops

- **Nested Loops -** A nested loop is a loop inside a loop.
- The "inner loop" will be executed one time for each iteration of the "outer loop":
- Example

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]
for x in adj:
    for y in fruits:
        print(x, y)
```

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

for loops

- **The pass Statement** - for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.
- Example

```
for x in [0, 1, 2]:
    pass
```

Special thanks to W3Schools

Class activity 5

เขียนคำสั่งภาษา Python เพื่อให้มีการทำงานของโปรแกรมดังนี้ ...

1. เขียน while loop สำหรับรับค่าตัวเลขหวยเลขท้าย 3 ตัว จากผู้ใช้งานจำนวน 10 ค่า เก็บลงใน list ที่ชื่อ lotto_list

2. จาก list คะแนนของนักเรียนจำนวน 5 คน score_list = [3.6, 5.5, 8.7, 9.9, 10.0] จงเขียน for loop เพื่อคำนวณหาค่าคะแนนเฉลี่ยของนักเรียนทั้ง 5 คนนี้

3. จาก list คะแนนของนักเรียนจำนวน 5 คน score_list = [3.6, 5.5, 8.7, 9.9, 10.0] จงเขียน for loop เพื่อคำนวณหาค่าคะแนนเฉลี่ยของนักเรียนที่ได้คะแนนต่ำกว่า 9 คะแนน