Lab 2: Program Control and Logic

Operators

Operators are used to perform operations on variables and values. Python has many operators for different purposes. The following are the commonly used operators.

Assignment Operator

We have already used the assignment operator =, which assigns the value of the right side operand to the left side variable. E.g.,

```
name = "Jack"
grade = 95
```

Arithmetic Operators

Arithmetic operators are used for performing mathematical operations. The following table shows the operators with some examples:

Name	Operator	Example
Addition	+	>>> 10 + 2 12
Subtraction	-	>>> 10 – 2 8
Multiplication	*	>>> 10 * 2 20
Division	/	>>> 10 / 2 5.0
Modulus	%	>>> 5 % 3 2
Exponentiation	**	>>> 10 ** 2 100
Floor division	//	>>> 5 // 3 1

Note: *An arithmetic* ***expression*** *can be constructed by using multiple arithmetic operators and parentheses.*

It is very common to assign an expression to a variable. E.g.,

```
PI = 3.14159
Radius = 6378
Perimeter = 2 * PI * Radius
print("The perimeter of earth is " + str(Perimeter))
Volume = 4/3 * PI * Radius ** 3
print("The volume of earth is {}".format(Volume))
```

Remark: The expression **4/3 * PI * Radius ** 3** is equivalent to **4/3 * PI * (Radius ** 3)** because ** has a higher priority (or precedence) than * and /. Now you can use Python as a calculator!

Comparison Operators (also called Relational Operators)

Comparison operators are used for comparing two values, and the value of a comparison expression is either True or False. The following table shows the operators with the examples:

Name	Operator	Example

Equal	==	>>> x = 1
		>>> x == 1
		True
		>>> x == 2
		False

Not Equal	!=	>>> x = 1
		>>> x != 1
		False
		>>> x != 2
		True

Greater than	>	>>> x = 1
		>>> x > 1
		False

Greater than or equal to	>=	>>> x = 1
		>>> x >= 1
		True

Less than	<	>>> x = 1
		>>> x < 1
		False

Less than or equal to	<=	>>> x = 1
		>>> x <= 1
		True

Logical Operators

Logical operators are used for combining the conditional statements. The following table shows the operators and the examples:

Operator	Description	Example

and	If both operands are True, then the result is True. Otherwise, the result if False.	>>> x = 5
		>>> x < 10 and x > 3
		True

or	If both operands are False, then the result if False. Otherwise, the result if True.	>>> x = 5
		>>> x == 1 or x == 5
		True

not	The result is the reverse of the operand.	>>> x = 5
		>>> not x > 10
		True

Control Statements

The most direct way to affect the flow of control is with a conditional statement. In Python, the main types of control statements are *if*, *while* and *for*.

if, if…else, if…elif…else statements

The *if* statement performs an indicated action only when the condition is true; otherwise the action is skipped. The syntax is:

```
if expression:
       statement
```

Remark: the colon : at the end of if statement in very important. It indicates that the following block of statements with the same amount of indentation will be executed if the expression is true.

Let's try the following example.

```
score = input('Please input the score: ')
score = float(score)

if score >= 50:
    print('Congratulations!')
    print('You passed the course!')
```

With the *else* clause (*if…else* statement), it allows you to specify that different actions are to be performed when the condition is false. The syntax is:

```
if expression:
       statement(s) 1
else:
       statement(s) 2
```

Try the following example.

```
score = input('Please input the score: ')
score = float(score)

if score >= 50:
    print('PASS')
    print('Congratulations!')
else:
    print('FAIL')
    print('Please work harder!')
```

The *if…elif…else* statement is used for multiple cases by placing the *elif* with conditions. The syntax is:

```
if expression 1:
       statement(s) 1
elif expression 2:
       statement(s) 2
elif …
       …
else
       statement(s) N
```

For example, the following code will print different grades based on the score.

```
score = input('Please input the score: ')
score = float(score)

if score >= 85:
    print('A')
elif score >= 75:
    print('B')
elif score >= 65:
    print('C')
elif score >= 50:
    print('D')
else:
    print('F')
```

while statement

The *while* statement is used to execute a block of statements repeatedly while some condition remains true. The syntax is:

```
while expression:
      statement(s)
```

Consider the following example code that prints the numbers from 1 to 10.

```
i = 1

while i <= 10:
    print(i)
    i = i+1

print(i)
```

The initial value of the variable *i* is 1. In the body of the while statement, *i* is printed and increased by 1. The loop runs total 10 times while *i* iterates from 1 to 10. Notice that after the while loop, i becomes 11.

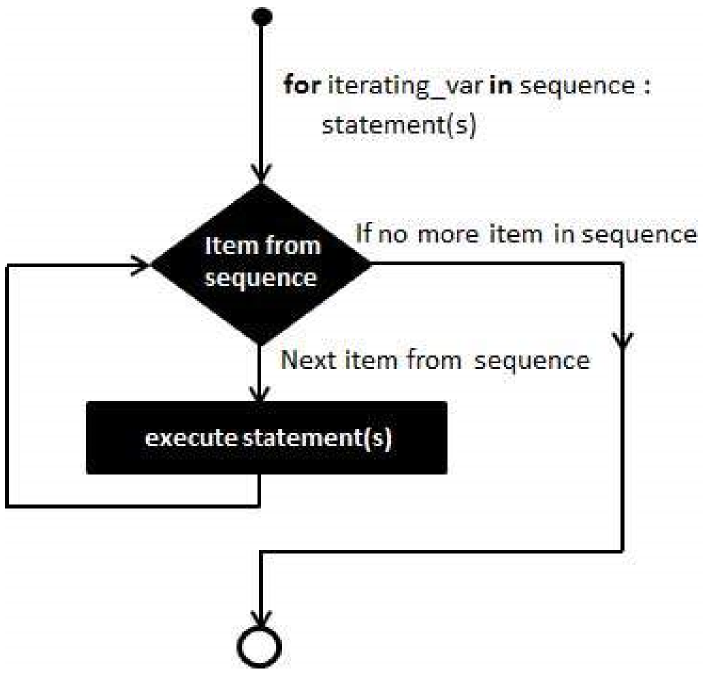
Try another example that uses while to implement a guess game:

```
answer = "Jack"
print("======Can you guess my name?======")
guess = ""
while guess != answer:
    guess = input("Guess my name: ")
print("Bingo!")
```

for statement

The *for* statement is used for iterating over a sequence. With the *for* statement, we can execute a block of statements, once for each item in the sequence (such as a list, tuple, range, and dictionary). The syntax is:

```
for iterating_var in sequence:
      statement(s)
```



Consider the following example code that prints the day names. The items in the list will be assigned to the iterating variable *d* one-by-one.

```
days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
'Sunday']
for d in days:
    print(d)
```

It is also very common to use the built-in function **range()** to generate an immutable sequence and then iterate through a sequence. For example:

```
days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
'Sunday']

for i in range(7):
    print(days[i])
```

break and *continue* statements

The *break* and *continue* statements are used to alter the flow of control when executed in a *while* or *for* statement.

The *break* statement causes an immediate exit from that statement. Therefore, the *break* statement is used to escape early from a loop.

Try the following code:

```
   #%%
1  for i in range(1, 10):
2      if i > 3:
3          break
4      print(i)
5  print('Broke out at x = {}'.format(i))
```

Output:

```
1
2
3
Broke out at x =  4
```

Remark: In the previous example, lines 2-4 are the statements of the for loop and they are indented by four spaces. Line 3, the break statement, is under the if statement; so it has another level of indentation of four spaces.

The *continue* statement skips the remaining statements in the body of that control statement and evaluate the condition immediately (*while* loop) or performs the next iteration (*for* loop).

Try the following code:

```
#%%
i = 0

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

Output:

```
1
2
4
5
6
```

Case Studies

Case Study 1 – Class Average Program 1

Imagine that a class of ten students took a quiz. The scores, integers in the range of [0, 100], for this quiz are available to us. We need to determine the grades and the class average on the quiz.

Assume that the scores are *50, 100, 75, 66, 88, 85, 46, 91, 0,* and *23*.

The grade assignment is based on the table below:

Score range	**Grade**
85 or above	A
75 to 84	B
65 to 74	C
50 to 64	D
Below 50	F

Let’s try the following code:

```
scores = [50, 100, 75, 66, 88, 85, 46, 91, 0, 23]
total = 0;

for s in scores:
    total += s

    if s >= 85:
        grade = 'A'
    elif s >= 75:
        grade = 'B'
    elif s >= 65:
        grade = 'C'
    elif s >= 50:
        grade = 'D'
    else:
        grade = 'F'

    print(i, grade, sep='\t')

print('Class average is ', total / len(scores))
```

Output:

```
50      D
100     A
75      B
66      C
88      A
85      A
46      F
91      A
0       F
23      F
Class average is 62.4
```

In this example, we use a *for* statement to get each score from the list and assign it to the variable *s*.

In the *for* loop, we use the operator **+=** to increase the value of *total* by *s*. It is exactly the same as **total = total + s**. So **+=** is also an assignment operator.

An *if…elif…else* statement is used for comparing the value of *s* and assign the grade.

The **print(i, grade, sep='\t')** statement outputs two values, *i* and *grade*, separated by a tab (i.e., **'\t'**).

Case Study 2 – Class Average Program 2

We need to implement a program for determining the grades and the class average on the quiz. Our program will prompt the user to input the scores until the input equals -1. The grade assignment is based on the same criteria mentioned in Case Study 1.

Let’s try the following code:

```
#%%
total = 0 # total marks
count = 0 # to count the number of students

while True:
    s = input('Enter score, -1 to end: ')
    s = int(s)

    if s < 0:
        break

    total += s
    count += 1
    if s >= 85:
        grade = 'A'
    elif s >= 75:
        grade = 'B'
    elif s >= 65:
        grade = 'C'
    elif s >= 50:
        grade = 'D'
    else:
        grade = 'F'

    print('Grade ', grade)

if count > 0:
    average = total / count
else:
    average = 0

print('There are {} students'.format(count))
print('Class average is {0:.2f}'.format(average))
```

Output:

```
Enter score, -1 to end: 75
Grade  B

Enter score, -1 to end: 45
Grade  F

Enter score, -1 to end: 63
Grade  D

Enter score, -1 to end: 15
Grade  F

Enter score, -1 to end: 82
Grade  B

Enter score, -1 to end: -1
There are 5 students
Class average is 56.00
```

As we do not know how many scores will be inputted, we use a while statement with **True**. It is an infinite loop that will not stop because the condition always is true.

In addition, we check whether the user input equals -1 or not. If yes, we use a **break** statement to exit the loop.

We use the variable **count** to count the number of scores inputted.

One more thing we need to pay attention to is that the user may input **-1** immediately after starting the program. The value of **count** will be zero. There is a zero division program when we calculate the average using the statement **average = total / count**. Therefore, we need to check whether **count** is greater than zero or not.

Finally, to only print 2 factional digits of variable ***average***, we use a placeholder {0:.2f} in the last **print** statement. The {0} indicates that the placeholder {} will be replaced by the 1st argument in the format(). The ".2f" indicates that the argument is a floating number with 2 digits after the period.

Case Study 3 – Shapes Printing Program

We are going to use Python program to print shapes with star symbols. The expected outputs are as follows.

```
*
**
***
****
*****
******
*******
********
*********
```

Let’s try the following code:

```
for i in range(10):
    for j in range(i):
        print('*', end='')
    print()
```

In the above example, we nest *for* statements inside another *for* statements. The outer for statement is used for printing ten rows. The inner for statement is used for printing a single row. Note that the **print()** function with **end=''** (empty string instead of the default ‘\n’ newline character) will print the value without a newline. And, the **print()** function without parameter will print a newline.

A more interesting example: In the following example, the program will output the same shape as an animation by sleeping for 0.5 second after displaying a *. Python has a **time** module that offers many useful functions related to time and date. To access this **time** module, we need to use **import time** first. The function **time.sleep(sec)** will suspend the execution of the program for the given number of seconds. We will learn more Python modules in this course.

```
import time # import the time module of Python

for i in range(10):
    for j in range(i):
        print('*', end='')
        time.sleep(0.5)        # pause for 0.5 second
    print()
```

Exercises

Exercise 1 – Greeting

Write a program to accept a user input and then print the greeting based on the following rules:

Rule	Input	Reply
1	“How are you?”	“Fine, and you?”
2	“How do you do.”	“How do you do.”
3	“Good to see you.”	“Me too.”
4	others	“…”

Hint:

- Use the function **input()** to get the user input.
- Use **If..then..else** statement to check the inputted text.
- Use the function **print()** to print the reply.
- The following is the sample output of the program:

```
Computer: Hi!
You: Good to see you.
Computer: Me too.
```

Exercise 2 – Summation

Write a program to accept integer inputs until the user inputs a non-integer. Then, the program prints the summation of the inputted numbers.

Hint:

- Use the function **input()** to get the user input.
- Use a variable **total** for calculating the summation.
- Need to use *while* loop for the repeated inputs.
- Use if statement with **isdigit()** function to check whether the user input is an integer or not.

```
if (n.isdigit() == False):
    break
```

- If the user input is an integer, use the function **int()** to convert the user input to an integer and add it to the variable **total**.

```
a = int(n)
```

- If the user input is not an integer, use the **break** statement to exit the loop immediately.
- The following is the sample output of the program:

```
Input an integer: 5

Input an integer: 10

Input an integer: 2

Input an integer: x
total =  17
```

Exercise 3 – Maximum Value

Write a program to get 5 integer inputs and then print the maximum one. Assume that the user will input integers only.

Hint:

- Use *for* statement to get user inputs 5 times.
- Use *if* statement to compare the integers.
- The following is the sample output of the program:

```
Input a number: 15

Input a number: 3

Input a number: 21

Input a number: 9

Input a number: 18
Maximum is  21
```

Please submit your solutions of the three exercises to Moodle.