

Fundamentals of Programming

Unit 3

Decision Making & Looping

Operators

How to perform our operations in Python?

Operators

Operator

a symbol which is responsible for a particular operation between two operands

□ Types of Operators in Python

- Arithmetic operators
- Comparison operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Arithmetic Operators

Arithmetic Operators

perform arithmetic operations between two operands

□ Types of Arithmetic Operators

Operator	Function	Description
+	Addition	Add values on either side of the operator
-	Subtraction	Subtract right hand operand from left hand operand
*	Multiplication	Multiply values on either side of the operator
/	Division	Divide left hand operand by right hand operand

Arithmetic Operators

□ Types of Arithmetic Operators

Operator	Function	Description
%	Modulus	Divide left hand operand by right hand operand and returns remainder
**	Exponent	Perform exponential (power) calculation on operators
//	Floor Division	The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity)

Arithmetic Operators

□ Example

- Input the following code and see the results

 *Example.py -

```
a = 20
b = 10
print("a + b = ", (a+b))
print("a - b = ", (a-b))
print("a * b = ", (a*b))
print("a / b = ", (a/b))
print("a % b = ", (a%b))

c = 5
d = 3
print("c ^ d = ", (c**d))
print("c // d = ", (c//d))
```

```
=====
a + b = 30
a - b = 10
a * b = 200
a / b = 2.0
a % b = 0
c ^ d = 125
c // d = 1
```

Programming Code

```
a = 20
b = 10
print("a + b = ", (a+b))
print("a - b = ", (a-b))
print("a * b = ", (a*b))
print("a / b = ", (a/b))
print("a % b = ", (a%b))

c = 5
d = 3
print("c ^ d = ", (c**d))
print("c // d = ", (c//d))
```

Arithmetic Operators

□ Class Activity

- Try to calculate the following equations

- $4 \times 7 + 10 \div 2$

- the remainder of $(5 \times 6 + 3) \div 7$

Assignment Operators

Assignment Operators

assign the value of the right expression to the left operand

□ Types of Assignment Operators

Operator	Description
=	Assign the value of the right expression to the left operand
+=	Increase the value of the left operand by the value of the right operand and assigns the modified value back to left operand
-=	Decrease the value of the left operand by the value of the right operand and assigns the modified value back to left operand

Assignment Operators

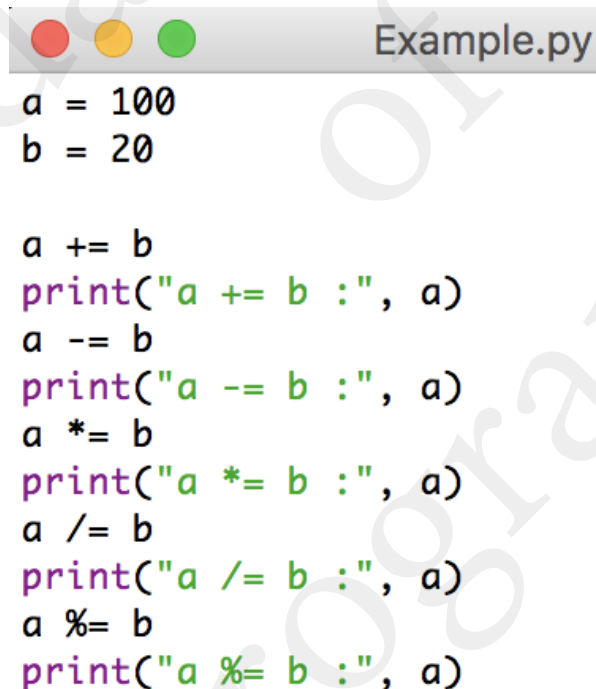
□ Types of Assignment Operators

Operator	Description
<code>*=</code>	multiply the value of the left operand by the value of the right operand and assigns the modified value back to then the left operand
<code>%=</code>	divide the value of the left operand by the value of the right operand and assigns the reminder back to the left operand
<code>**=</code>	<code>a**=b</code> will be equal to <code>a=a**b</code>
<code>//=</code>	<code>a//=b</code> will be equal to <code>a = a// b</code>

Assignment Operators

□ Example

- Input the following code and see the results



```
a = 100
b = 20

a += b
print("a += b :", a)
a -= b
print("a -= b :", a)
a *= b
print("a *= b :", a)
a /= b
print("a /= b :", a)
a %= b
print("a %= b :", a)
```

Programming Code

```
a = 100
b = 20

a += b
print("a += b :", a)
a -= b
print("a -= b :", a)
a *= b
print("a *= b :", a)
a /= b
print("a /= b :", a)
a %= b
print("a %= b :", a)
```

Assignment Operators

□ Example

- Input the following code and see the results



```
c = 2
d = 3
c **= d
print("c **= d :", c)
```

```
e = 10
f = 3
e //= f
print("e //= f :", e)
```

Programming Code

```
c = 2
d = 3
c **= d
print("c **= d :", c)
```

```
e = 10
f = 3
e //= f
print("e //= f :", e)
```

Operator Precedence

□ Operator Precedence

- precedence of the operators is essential to find out since it enables us to know which operator should be evaluated first

Operator	Description
**	The exponent operator is given priority over all the others used in the expression
~ + -	The negation, unary plus, and minus
* / % //	The multiplication, divide, modules, reminder, and floor division
+ -	Binary plus, and minus
>> <<	Left shift. and right shift
&	Binary and.
^	Binary xor, and or
<= < > >=	Comparison operators
<> == !=	Equality operators.
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Decision Making

Can we do different operations based on different conditions?

Decision Making

Decision Making

anticipation of conditions occurring while execution of the program
and specifying actions taken according to the conditions

□ General Working Mechanisms

- decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome
- determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise

Decision Making

❑ True or False?

- TRUE → any non-zero and non-null values
- FALSE → either zero or null

❑ Types of Decision Making Statements

1. if statements
2. if...else statements
3. nested if statements

Decision Making

□ Indentation in Decision Making

- Python doesn't allow the use of parentheses for the block level code
- indentation is used to declare a block
- four spaces are given to indent the statements which are a typical amount of indentation in Python generally
- all the statements of one block are intended at the same level indentation

if Statement

if Statement

an *if* statement consists of a boolean expression followed by one or more statements


□ General Working Mechanisms

- if the boolean expression evaluates to **TRUE**
 - the block of statement(s) inside the if statement is executed
- if boolean expression evaluates to **FALSE**
 - the first set of code after the end of the if statement(s) is executed

if Statement

□ Example

- Input the following code and see the results



```
a = 10
if (a < 100):
    print("a is smaller than 100")
```

Programming Code

```
a = 10
if (a < 100):
    print("a is smaller than 100")
```

Comparison Operators

Comparison Operators

compare the value of the two operands and returns Boolean true or false accordingly

□ Types of Comparison Operators

Operator	Description
==	If the values of two operands are equal, then the condition becomes true
!=	If values of two operands are not equal, then condition becomes true
<>	If values of two operands are not equal, then condition becomes true

Comparison Operators

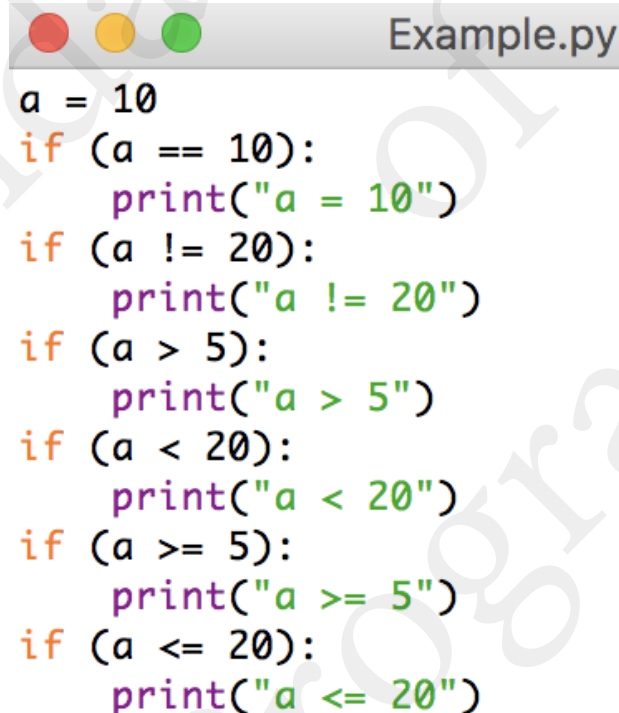
□ Types of Comparison Operators

Operator	Description
>	If the value of left operand is greater than the value of right operand, then condition becomes true
<	If the value of left operand is less than the value of right operand, then condition becomes true
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true

Comparison Operators

□ Example

- Input the following code and see the results



```
a = 10
if (a == 10):
    print("a = 10")
if (a != 20):
    print("a != 20")
if (a > 5):
    print("a > 5")
if (a < 20):
    print("a < 20")
if (a >= 5):
    print("a >= 5")
if (a <= 20):
    print("a <= 20")
```

Programming Code

```
a = 10
if (a == 10):
    print("a = 10")
if (a != 20):
    print("a != 20")
if (a > 5):
    print("a > 5")
if (a < 20):
    print("a < 20")
if (a >= 5):
    print("a >= 5")
if (a <= 20):
    print("a <= 20")
```

Comparison Operators

❑ Class Activity

- Ask the user to input the score and store it into a variable named **score** and check for the following conditions. Sample output is shown below.
 1. if the score is greater or equal to 50, then the student can pass this course
 2. If the score is less than 50, then the student needs to retake this course
 3. If the score is greater than 90, then the student gets the distinction

Sample Output 1

```
What is your score? 55.5  
You get a pass
```

Sample Output 2

```
What is your score? 45  
You need to retake this course
```

Sample Output 3

```
You get a pass  
You get a distinction
```

Logical Operators

Logical Operators

primarily in the expression evaluation to make a decision

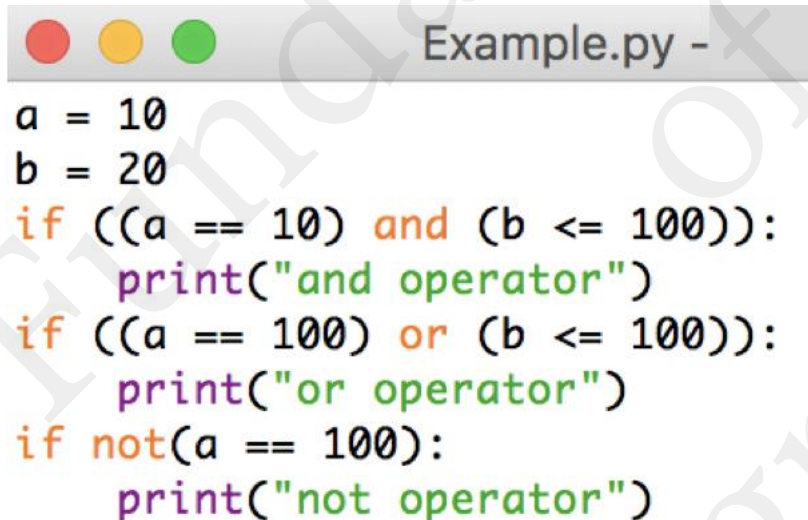
□ Types of Logical Operators

Operator	Description
and	If both the expression are true, then the condition will be true
or	If one of the expressions is true, then the condition will be true
not	If an expression a is true, then not (a) will be false and vice versa

Logical Operators

❑ Example

- Input the following code and see the results



```
a = 10
b = 20
if ((a == 10) and (b <= 100)):
    print("and operator")
if ((a == 100) or (b <= 100)):
    print("or operator")
if not(a == 100):
    print("not operator")
```

Programming Code

```
a = 10
b = 20
if ((a == 10) and (b <= 100)):
    print("and operator")
if ((a == 100) or (b <= 100)):
    print("or operator")
if not(a == 100):
    print("not operator")
```


Logical Operators

□ Class Activity

- Ask the user to input the arts and business courses, check for the following conditions and display the output as shown
 1. If a student got 90 marks or above in both courses, then the student can get the scholarship
 2. If a student got 90 marks or above in one of the courses, and both of the course have the scores higher than or equal to 50, the student can get a “A” grade
 3. if none of the course is below 50 marks, then the student can get the certificate

Logical Operators

❑ Class Activity

Sample Output 1

```
Enter the Art Score: 95.5
Enter the Business Score: 92
-----Result-----
You can get the scholarship
You can get a "A" grade
You can get the certificate
```

Sample Output 3

```
Enter the Art Score: 85
Enter the Business Score: 87.5
-----Result-----
You can get the certificate
```

Sample Output 2

```
Enter the Art Score: 95
Enter the Business Score: 85.5
-----Result-----
You can get a "A" grade
You can get the certificate
```

Sample Output 4

```
Enter the Art Score: 45
Enter the Business Score: 90.5
-----Result-----
```

if...else Statements

if...else Statements

an *if* statement can be followed by an optional ***else*** statement, which executes when the boolean expression is FALSE

□ General Working Mechanisms

- an ***else*** statement contains the block of code that executes if the conditional expression in the *if* statement resolves to 0 or a **FALSE** value
- ***else*** statement is an optional statement → at most only one else statement following *if*

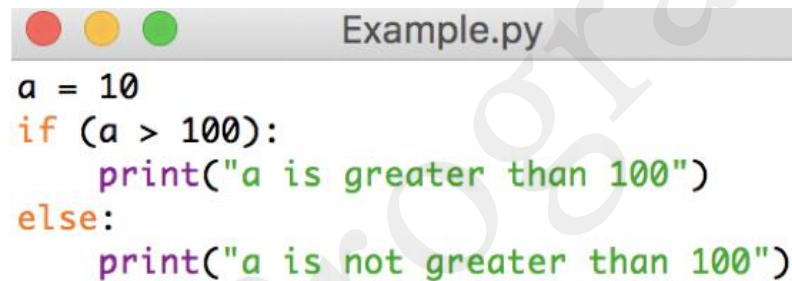
if...else Statements

□ Example

- Input the following code and see the results

Programming Code

```
a = 10
if (a > 100):
    print("a is greater than 100")
else:
    print("a is not greater than 100")
```



```
Example.py
a = 10
if (a > 100):
    print("a is greater than 100")
else:
    print("a is not greater than 100")
```

if...else Statements

□ Class Activity

- Ask the user for the current temperature. Check for the following conditions using if...else statement and display the output as shown.
 1. If the temperature is higher than 30 degrees, then the weather is hot.
 2. Otherwise, the weather is cool.

Sample Output 1

```
Current temperature: 35.5  
The weather is hot
```

Sample Output 2

```
Current temperature: 18  
The weather is cool
```

elif Statements

elif Statements

check multiple conditions and execute the specific block of statements depending upon the true condition among them

□ General Working Mechanisms

- check multiple expressions for **TRUE** and execute a block of code as soon as one of the conditions evaluates to **TRUE**
- similar to the *else*, the *elif* statement is optional
- there can be an arbitrary number of *elif* statements following an *if*

elif Statements

□ Syntax

```
if expression1:  
    statement(s)  
elif expression2:  
    statement(s)  
elif expression3:  
    statement(s)  
else:  
    statement(s)
```

□ Features

- core Python does not provide switch or case statements as in other languages
- can use *if..elif...* statements to simulate switch case

elif Statements

□ Example

- Input the following code and see the results



```
a = 10
```

```
if (a <= 0):  
    print("a <= 0")  
elif (a <= 5):  
    print("a <= 5")  
elif (a <= 10):  
    print("a <= 10")  
else:  
    print("a >10")
```

Programming Code

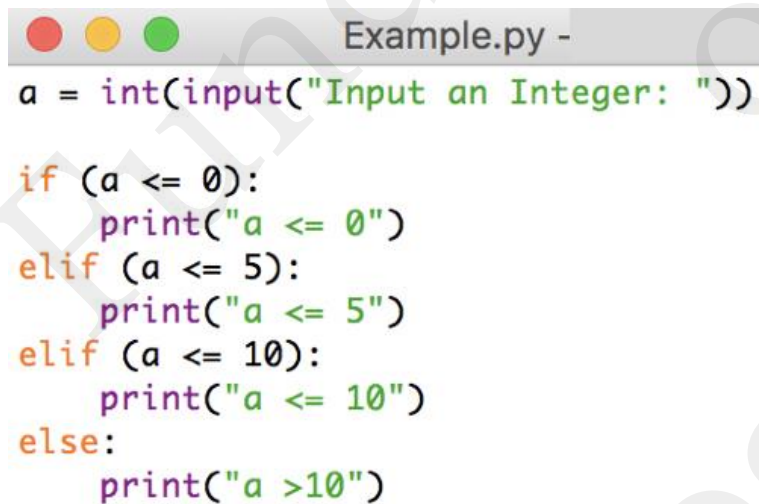
```
a = 10
```

```
if (a <= 0):  
    print("a <= 0")  
elif (a <= 5):  
    print("a <= 5")  
elif (a <= 10):  
    print("a <= 10")  
else:  
    print("a >10")
```


elif Statements

□ Example

- Input the following code and see the results



```
a = int(input("Input an Integer: "))

if (a <= 0):
    print("a <= 0")
elif (a <= 5):
    print("a <= 5")
elif (a <= 10):
    print("a <= 10")
else:
    print("a >10")
```

Programming Code

```
a = int(input("Input an Integer: "))

if (a <= 0):
    print("a <= 0")
elif (a <= 5):
    print("a <= 5")
elif (a <= 10):
    print("a <= 10")
else:
    print("a >10")
```

elif Statements

❑ Class Activity

- Ask the user for the current temperature. Check for the following conditions using elif statement and display the output as shown.
 1. If the temperature is 30 degrees above, then the weather is hot
 2. If the temperature is between 15 degrees and 30 degrees, then the weather is cool
 3. Otherwise, the temperature is cold

Sample Output 1

```
Current temperature: 34  
The weather is hot
```

Sample Output 2

```
Current temperature: 18.5  
The weather is cool
```

Sample Output 3

```
Current temperature: 12  
The weather is cold
```

Nested if Statements

Nested if Statements

use one *if* or *else if* statement inside another *if* or *else if* statement(s)

□ General Working Mechanisms

- check multiple expressions for **TRUE** and execute a block of code as soon as one of the conditions evaluates to **TRUE**
- similar to the *else*, the *elif* statement is optional
- there can be an arbitrary number of *elif* statements following an *if*

Nested if Statements

□ Syntax

```
if expression1:  
    statement(s)  
    if expression2:  
        statement(s)  
    elif expression3:  
        statement(s)  
    elif expression4:  
        statement(s)  
    else:  
        statement(s)  
else:  
    statement(s)
```

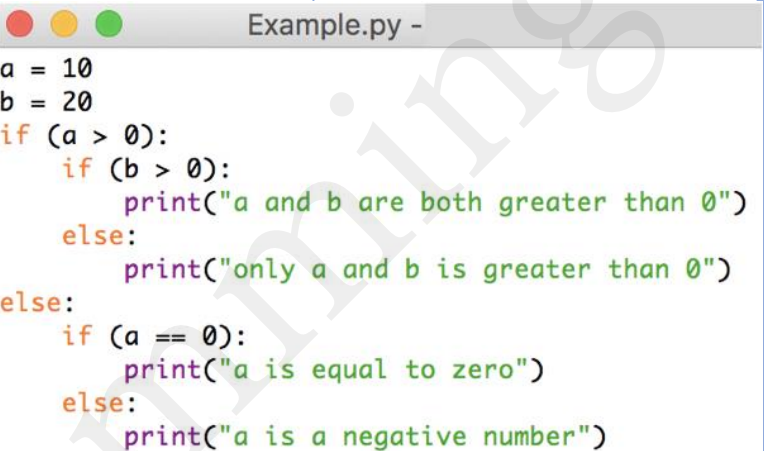
Nested if Statements

□ Example

- Input the following code and see the results

Programming Code

```
a = 10
b = 20
if (a > 0):
    if (b > 0):
        print("a and b are both greater than 0")
    else:
        print("only a and b is greater than 0")
else:
    if (a == 0):
        print("a is equal to zero")
    else:
        print("a is a negative number")
```



```
Example.py -
a = 10
b = 20
if (a > 0):
    if (b > 0):
        print("a and b are both greater than 0")
    else:
        print("only a and b is greater than 0")
else:
    if (a == 0):
        print("a is equal to zero")
    else:
        print("a is a negative number")
```

Nested if Statements

❑ Class Activity

- Write a python program for a cinema with the following conditions shown in the table using nested if statements. Ask the user to input the number of tickets they want to buy and whether they are members and/or students.
- Assume the ticket price is \$50 and user will only input “Y” and “N” for answering the questions of whether they are members and/or students. Display the output as shown.

Member	Student	Discount / Promotion
✓	✓	• 20% off of the ticket price
✓	X	• 10% off of the ticket price
X	✓	• 5% off of the ticket price • Get a \$10 coupon
X	X	• No discount

Nested if Statements

□ Class Activity

Sample Output 1

```
How many tickets you want to buy? 5
Are you the member? (Y/N) Y
Are you a student? (Y/N) Y
Original price: 250.0
Total price: 200.0
```

Sample Output 2

```
How many tickets you want to buy? 5
Are you the member? (Y/N) Y
Are you a student? (Y/N) N
Original price: 250.0
Total price: 225.0
```

Sample Output 3

```
How many tickets you want to buy? 5
Are you the member? (Y/N) N
Are you a student? (Y/N) Y
Original price: 250.0
Total price: 237.5
You can get a $10 coupon
```

Sample Output 4

```
How many tickets you want to buy? 5
Are you the member? (Y/N) N
Are you a student? (Y/N) N
Original price: 250.0
Sorry No discount.
```

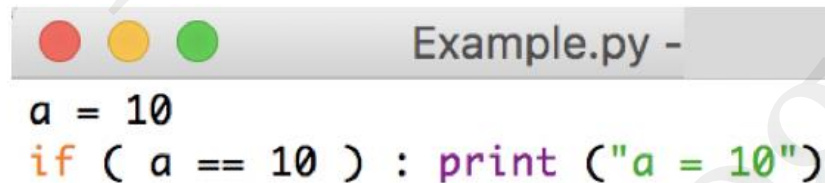
Single Statement Suites

Single Statement Suites

If the suite of an *if* clause consists only of a single line, it may go on the same line as the header statement

□ Example

- Input the following code and see the results



```
a = 10
if ( a == 10 ) : print ( "a = 10" )
```

Programming Code

```
a = 10
if ( a == 10 ) : print ( "a = 10" )
```


Looping

Perform repetitive operations ...

Looping

Looping

execute a statement or group of statements multiple times

□ Background

- statements are executed sequentially → The first statement in a function is executed first, followed by the second, and so on
- there may be a situation when you need to execute a block of code several number of times
- programming languages provide various control structures that allow for more complicated execution paths

Looping

❑ Types of Looping

1. while loop
2. for loop
3. nested loop

❑ Advantages of Using Loops

- code re-usability
- do not need to write the same code again and again
- traverse over the elements of data structures (array or linked lists)

while Loop

while Loop

repeat a statement or group of statements while a given condition is **TRUE**
test the condition before executing the loop body

□ General Working Mechanisms

- statement(s) may be a single statement or a block of statements
- the condition may be any expression, and **TRUE** is any non-zero value → the loop iterates while the condition is **TRUE**
- when the condition becomes **FALSE**, program control passes to the line immediately following the loop

while Loop

□ Example

- Input the following code and see the results



```
a = 0
while (a <= 5):
    print(a)
    a = a + 1
```

Programming Code

```
a = 0
while (a <= 5):
    print(a)
    a = a + 1
```

while Loop

❑ Class Activity

- Print the following output using **while** loop



```
a = 20
while (a <= 30):
    print(a)
    a = a + 2
```

Programming Code

```
a = 20
while (a <= 30):
    print(a)
    a = a + 2
```

Infinite Loop

□ Infinite Loop

- a loop becomes infinite loop if a condition never becomes FALSE
- the possibility that this condition never resolves to a FALSE value
- a loop that never ends → an infinite loop
- an infinite loop might be useful in client/server programming where the server needs to run continuously so that client programs can communicate with it as and when required

Infinite Loop

❑ Example

- Input the following code and see the results



```
a = 0
while (a <= 5):
    print(a)
```

Programming Code

```
a = 0
while (a <= 5):
    print(a)
```


while Loop

❑ Use with *else* Statement

- Python supports to have an *else* statement associated with a loop statement
- If the *else* statement is used with a *while* loop, the *else* statement is executed when the condition becomes **FALSE**

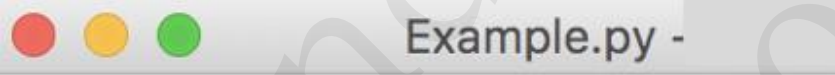
❑ Single Statement Suites

- similar to the if statement syntax, if your *while* clause consists only of a single statement, it may be placed on the same line as the while header

while Loop

□ Example

- Print the following output using **while** loop



```
a = 0
while (a <= 5):
    print(a)
    a = a + 1
else:
    print("a is greater than 5")
```

Programming Code

```
a = 0
```

```
while (a <= 5):
```

```
    print(a)
```

```
    a = a + 1
```

```
else:
```

```
    print("a is greater than 5")
```

while Loop

❑ Class Activity

- Print the following output using *while* loop

===== RESTART:

You are having 1 lesson

You are having 2 lesson

You are having 3 lesson

You are having 4 lesson

All lessons are completed

for Loop

for Loop

execute a sequence of statements multiple times and abbreviates the code that manages the loop variable

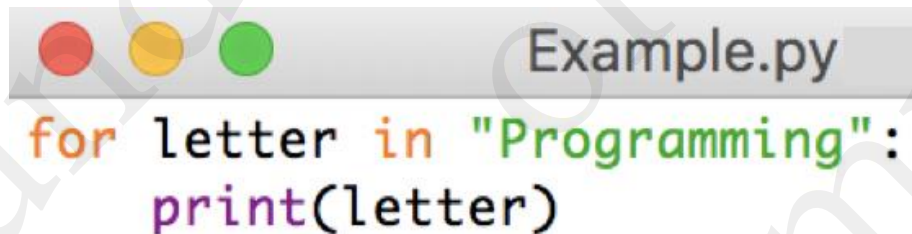
□ General Working Mechanisms

- if a sequence contains an expression list, it is evaluated first
- the first item in the sequence is assigned to the iterating variable
- the statements block is executed
- each item in the list is assigned to iterating variable, and the statement(s) block is executed until the entire sequence is exhausted

for Loop

□ Example

- Input the following code and see the results



```
for letter in "Programming":  
    print(letter)
```

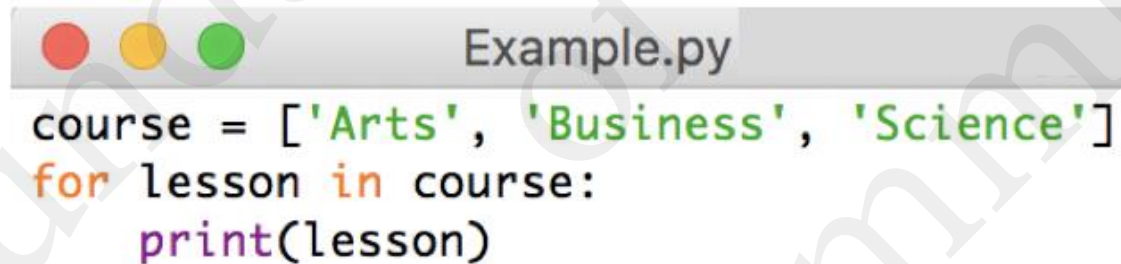
Programming Code

```
for letter in "Programming":  
    print(letter)
```

for Loop

□ Example

- Input the following code and see the results



```
course = ['Arts', 'Business', 'Science']  
for lesson in course:  
    print(lesson)
```

Programming Code

```
course = ['Arts', 'Business', 'Science']  
for lesson in course:  
    print(lesson)
```

for Loop

❑ Iterating by Sequence Index

- an alternative way of iterating through each item is by index offset into the sequence itself
- the ***len()*** built-in function → provide the total number of elements in the tuple as well as the ***range()*** built-in function to give us the actual sequence to iterate over

❑ Use with ***else*** Statement

- if the ***else*** statement is used with a ***for*** loop, the else statement is executed when the loop has exhausted iterating the list

for Loop

□ Example

- Input the following code and see the results



```
for a in range(0, 4):  
    print(a)
```

Programming Code

```
for a in range(0, 4):  
    print(a)
```


for Loop

❑ Class Activity

- Print the following output using *for* loop

```
===== RESTART
Number of Inventory: 10
Number of Inventory: 11
Number of Inventory: 12
Number of Inventory: 13
Number of Inventory: 14
```

Nested Loops

❑ Nested Loop

- Python programming language allows to use one loop inside another loop
- we can put any type of loop inside of any other type of loop

❑ Syntax

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

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

Nested Loops

□ Example

- Input the following code and see the results

Programming Code

```
course = ['Arts', 'Business', 'Science']
```

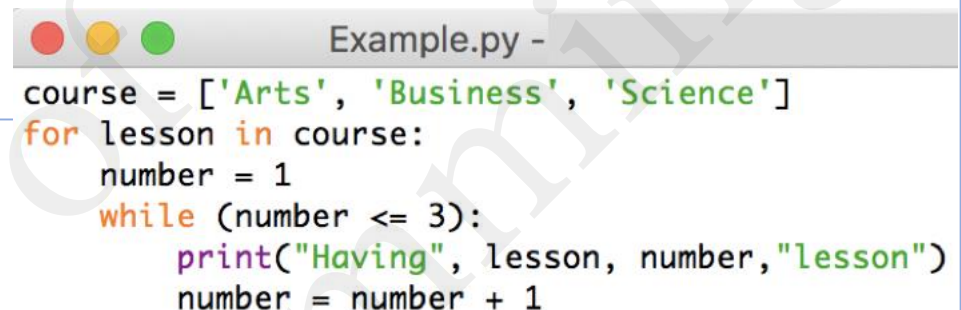
```
for lesson in course:
```

```
    number = 1
```

```
    while (number <= 3):
```

```
        print("Having", lesson, number, "lesson")
```

```
        number = number + 1
```



```
course = ['Arts', 'Business', 'Science']
for lesson in course:
    number = 1
    while (number <= 3):
        print("Having", lesson, number, "lesson")
        number = number + 1
```

Nested Loops

□ Class Activity

- Print the following output using nested loops

```
=====
10 x 1 = 10
10 x 2 = 20
10 x 3 = 30
10 x 4 = 40
11 x 1 = 11
11 x 2 = 22
11 x 3 = 33
11 x 4 = 44
12 x 1 = 12
12 x 2 = 24
12 x 3 = 36
12 x 4 = 48
13 x 1 = 13
13 x 2 = 26
13 x 3 = 39
13 x 4 = 52
```

Hope you enjoy the class

Thank you