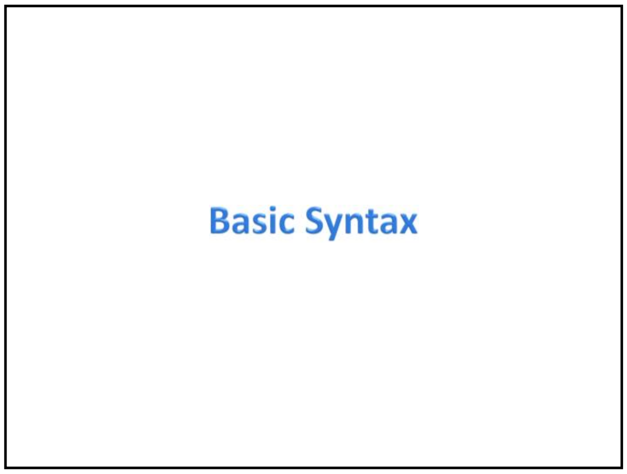
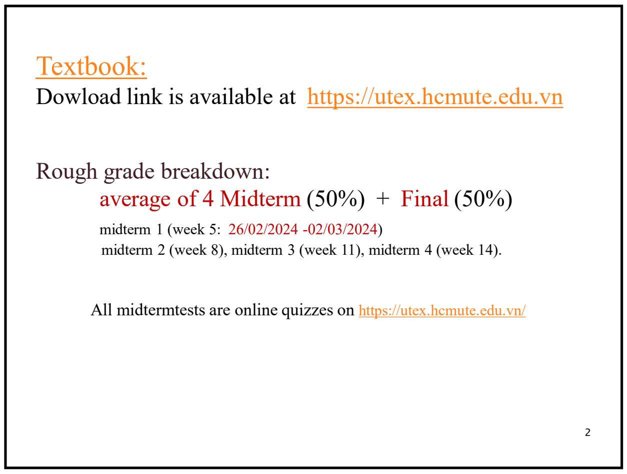


Basic Syntax

1



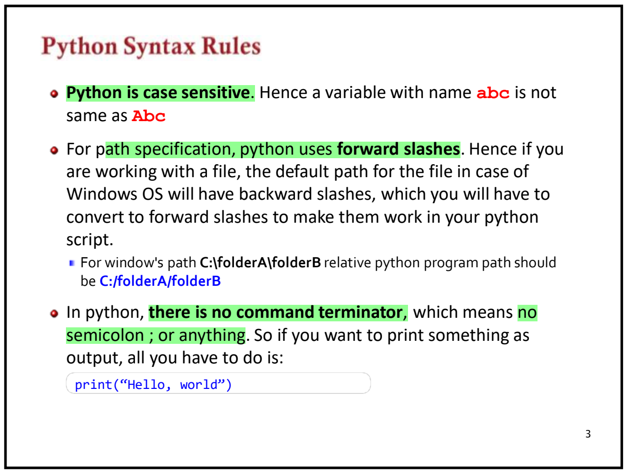
Textbook:
Dowload link is available at https://utex.hcmute.edu.vn

Rough grade breakdown:
average of 4 Midterm (50%) + Final (50%)
midterm 1 (week 5: 26/02/2024 -02/03/2024)
midterm 2 (week 8), midterm 3 (week 11), midterm 4 (week 14).

All midtermtests are online quizzes on https://utex.hcmute.edu.vn/

2

2



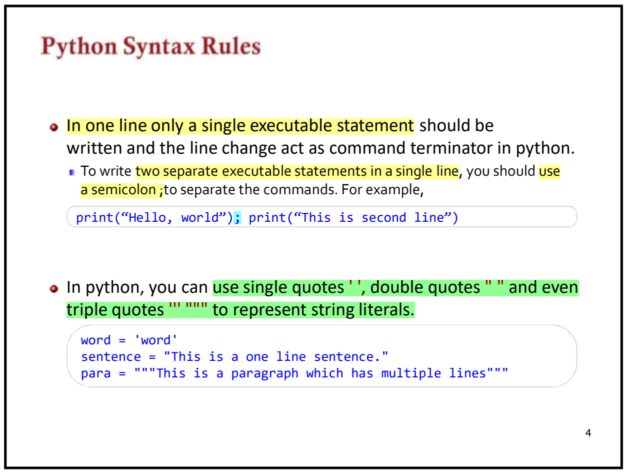
Python Syntax Rules

- Python is case sensitive. Hence a variable with name **abc** is not same as **Abc**
- For path specification, python uses **forward slashes**. Hence if you are working with a file, the default path for the file in case of Windows OS will have backward slashes, which you will have to convert to forward slashes to make them work in your python script.
  - For window's path **C:\folderA\folderB** relative python program path should be **C:/folderA/folderB**
- In python, **there is no command terminator,** which means no semicolon ; or anything. So if you want to print something as output, all you have to do is:

```
print("Hello, world")
```

3

3



Python Syntax Rules

- In one line only a single executable statement should be written and the line change act as command terminator in python.
  - To write two separate executable statements in a single line, you should use a semicolon ;to separate the commands. For example,

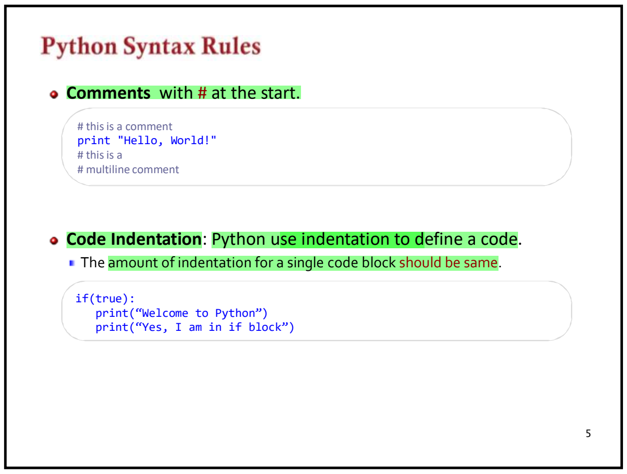
```
print("Hello, world"); print("This is second line")
```

- In python, you can use single quotes ' ', double quotes " " and even triple quotes ''' """ to represent string literals.

```
word = 'word'
sentence = "This is a one line sentence."
para = """This is a paragraph which has multiple lines"""
```

4

4



Python Syntax Rules

- **Comments** with # at the start.

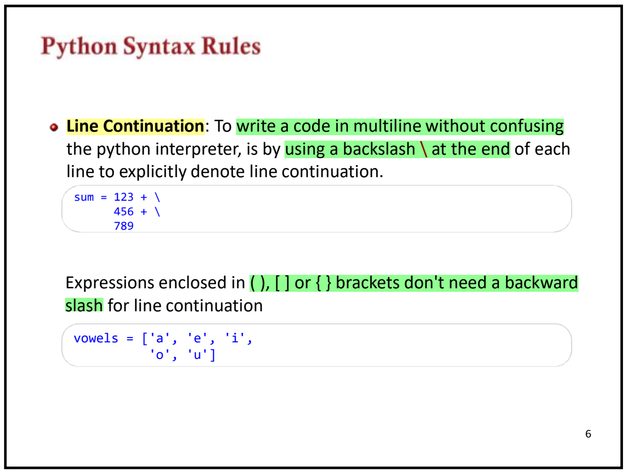
```
# this is a comment
print "Hello, World!"
# this is a
# multiline comment
```

- **Code Indentation**: Python use indentation to define a code.
  - The amount of indentation for a single code block should be same.

```
if(true):
   print("Welcome to Python")
   print("Yes, I am in if block")
```

5

5



Python Syntax Rules

- **Line Continuation**: To write a code in multiline without confusing the python interpreter, is by using a backslash \ at the end of each line to explicitly denote line continuation.

```
sum = 123 + \
      456 + \
      789
```

Expressions enclosed in ( ), [ ] or { } brackets don't need a backward slash for line continuation

```
vowels = ['a', 'e', 'i',
          'o', 'u']
```

6

6

First Python Program

- To print **I am a student** on screen, all you have to do is:

```
print("I am a student")
```

- You can write and execute this code in IDLE, or you can save this code in a python code file, name it **code.py** (you can name it anything, just keep the extension of the file as .py). To run the **code.py** python script, open IDLE, go to the directory where you saved this file, and then type the following in command prompt or your terminal: `python code.py`

Identifiers

- An identifier is a name given to a variable, function, class or module.
- Format:
  - Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_). Names like myCountry, other_1 and good_morning, all are valid examples. A Python identifier can begin with an alphabet (A – Z and a – z and _).
  - An identifier cannot start with a digit but is allowed everywhere else. 1plus is invalid, but plus1 is perfectly fine.
  - Keywords cannot be used as identifiers.
  - One cannot use spaces and special symbols like !, @, #, $, % etc. as identifiers.
  - Identifier can be of any length

Keywords

List of Keywords in Python

and	as	not
assert	finally	or
break	for	pass
class	from	nonlocal
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield
False	True	None

- Attempting to use a keyword as an identifier name will cause an error

Variables

- Rules for creating Legal Variable Names
  - Variable names can consist of any number of letters, underscores and digits.
  - Variable should not start with a number.
  - Python Keywords are not allowed as variable names.
  - Variable names are case-sensitive.
- Assigning Values to Variables
  - Format: *variable_name = expression*
  - Python allows you to assign a single value to several variables simultaneously

```
x = 12
y = 2.13
s = 'hello'
a = b = c = 1
```

Operators

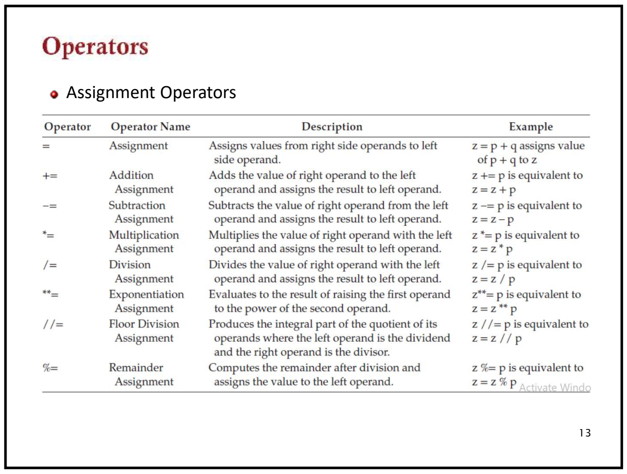
- Operators in Python
  - Arithmetic Operators
  - Assignment Operators
  - Comparison Operators
  - Logical Operators
  - Bitwise Operators

Operators

- Arithmetic Operators

Operator	Operator Name	Description	Example
+	Addition operator	Adds two operands, producing their sum.	p + q = 5
−	Subtraction operator	Subtracts the two operands, producing their difference.	p − q = −1
*	Multiplication operator	Produces the product of the operands.	p * q = 6
/	Division operator	Produces the quotient of its operands where the left operand is the dividend and the right operand is the divisor.	q / p = 1.5
%	Modulus operator	Divides left hand operand by right hand operand and returns a remainder.	q % p = 1
**	Exponent operator	Performs exponential (power) calculation on operators.	p**q = 8
//	Floor division operator	Returns the integral part of the quotient.	9//2 = 4 and 9.0//2.0 = 4.0

Note: The value of p is 2 and q is 3.



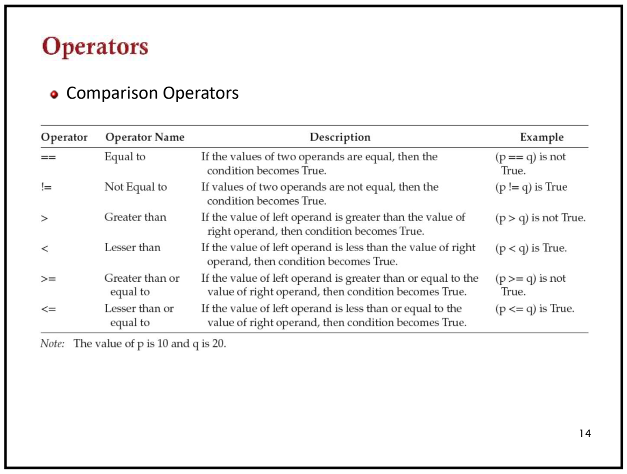
Operators

- Assignment Operators

Operator	Operator Name	Description	Example
=	Assignment	Assigns values from right side operands to left side operand.	z = p + q assigns value of p + q to z
+=	Addition Assignment	Adds the value of right operand to the left operand and assigns the result to left operand.	z += p is equivalent to z = z + p
−=	Subtraction Assignment	Subtracts the value of right operand from the left operand and assigns the result to left operand.	z −= p is equivalent to z = z − p
*=	Multiplication Assignment	Multiplies the value of right operand with the left operand and assigns the result to left operand.	z *= p is equivalent to z = z * p
/=	Division Assignment	Divides the value of right operand with the left operand and assigns the result to left operand.	z /= p is equivalent to z = z / p
**=	Exponentiation Assignment	Evaluates to the result of raising the first operand to the power of the second operand.	z**= p is equivalent to z = z ** p
//=	Floor Division Assignment	Produces the integral part of the quotient of its operands where the left operand is the dividend and the right operand is the divisor.	z //= p is equivalent to z = z // p
%=	Remainder Assignment	Computes the remainder after division and assigns the value to the left operand.	z %= p is equivalent to z = z % p

13

13



Operators

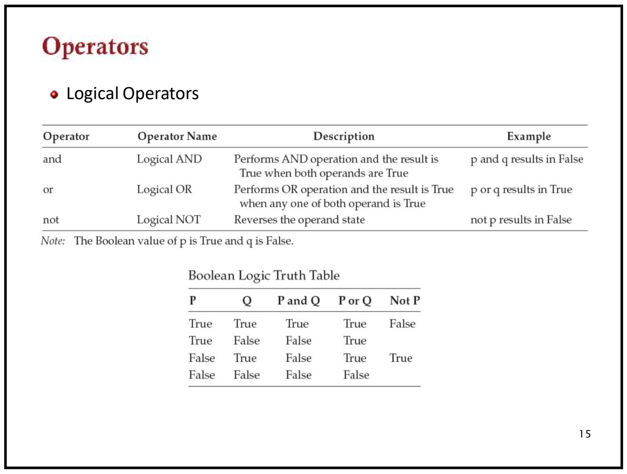
- Comparison Operators

Operator	Operator Name	Description	Example
==	Equal to	If the values of two operands are equal, then the condition becomes True.	(p == q) is not True.
!=	Not Equal to	If values of two operands are not equal, then the condition becomes True.	(p != q) is True
>	Greater than	If the value of left operand is greater than the value of right operand, then condition becomes True.	(p > q) is not True.
<	Lesser than	If the value of left operand is less than the value of right operand, then condition becomes True.	(p < q) is True.
>=	Greater than or equal to	If the value of left operand is greater than or equal to the value of right operand, then condition becomes True.	(p >= q) is not True.
<=	Lesser than or equal to	If the value of left operand is less than or equal to the value of right operand, then condition becomes True.	(p <= q) is True.

Note: The value of p is 10 and q is 20.

14

14



Operators

- Logical Operators

Operator	Operator Name	Description	Example
and	Logical AND	Performs AND operation and the result is True when both operands are True	p and q results in False
or	Logical OR	Performs OR operation and the result is True when any one of both operand is True	p or q results in True
not	Logical NOT	Reverses the operand state	not p results in False

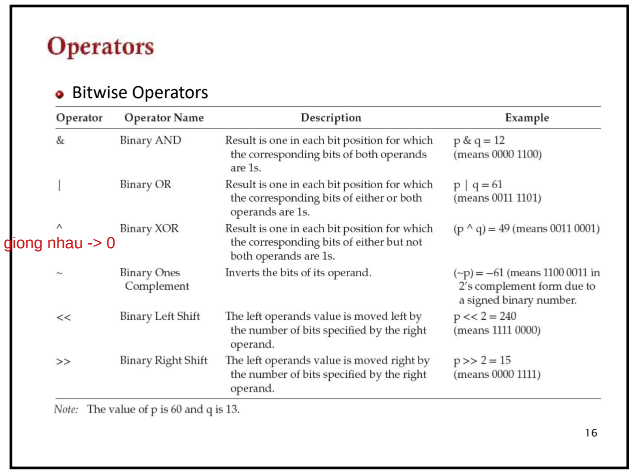
Note: The Boolean value of p is True and q is False.

Boolean Logic Truth Table

P	Q	P and Q	P or Q	Not P
True	True	True	True	False
True	False	False	True	
False	True	False	True	True
False	False	False	False	

15

15



Operators

- Bitwise Operators

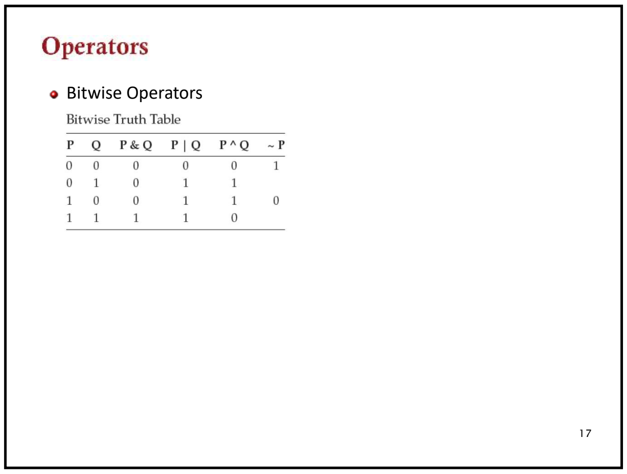
Operator	Operator Name	Description	Example
&	Binary AND	Result is one in each bit position for which the corresponding bits of both operands are 1s.	p & q = 12 (means 0000 1100)
|	Binary OR	Result is one in each bit position for which the corresponding bits of either or both operands are 1s.	p | q = 61 (means 0011 1101)
^	Binary XOR	Result is one in each bit position for which the corresponding bits of either but not both operands are 1s.	(p ^ q) = 49 (means 0011 0001)
~	Binary Ones Complement	Inverts the bits of its operand.	(~p) = −61 (means 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	p << 2 = 240 (means 1111 0000)
>>	Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	p >> 2 = 15 (means 0000 1111)

xor: 2 bit giong nhau -> 0

Note: The value of p is 60 and q is 13.

16

16



Operators

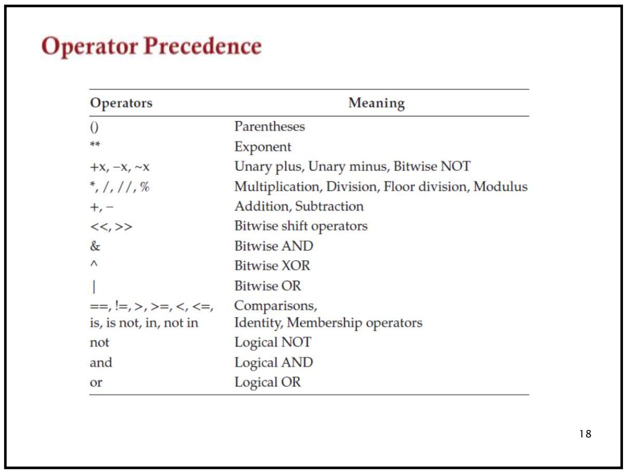
- Bitwise Operators

Bitwise Truth Table

P	Q	P & Q	P | Q	P ^ Q	~ P
0	0	0	0	0	1
0	1	0	1	1	
1	0	0	1	1	0
1	1	1	1	0	

17

17



Operator Precedence

Operators	Meaning
()	Parentheses
**	Exponent
+x, −x, ~x	Unary plus, Unary minus, Bitwise NOT
*, /, //, %	Multiplication, Division, Floor division, Modulus
+, −	Addition, Subtraction
<<, >>	Bitwise shift operators
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
==, !=, >, >=, <, <=, is, is not, in, not in	Comparisons, Identity, Membership operators
not	Logical NOT
and	Logical AND
or	Logical OR

18

18

Data Types

- Basic data types of Python are
  - Numbers:
    - `int.` Integers can be of any length; it is only limited by the memory available
    - `Float` A floating point number is accurate up to 15 decimal places
    - `Complex` Complex numbers are written in the form, x + yj, where x is the real part and y is the imaginary part.
  - Boolean
    - `Boolean values:` **True** and **False** are treated as reserved words
  - Strings
    - `A string literal is zero or more characters enclosed in double (") or single (') quotation marks.`
    - `Multiline strings can be denoted using triple quotes, ''' or """`
  - None
    - `None is another special data type in Python. None is frequently used to represent the absence of a value.`

19

19

Indentation

- In python, if a code block has to be deeply nested, then the nested statements need to be indented further to the right

20

20

Comments

- Single Line Comment
  - In Python, use the hash (#) symbol to start writing a comment. For example,

```
#This is single line Python comment
```

- Multiline Comments
  - If the comment extends multiple lines, then one way of commenting those lines is to use hash (#) symbol at the beginning of each line. For example,

```
#This is
#multiline comment
#in Python
```

  - Another way of doing this is to use triple quotes, either ''' or """. These triple quotes are generally used for multiline strings.

```
'''This is
   multiline comment
   in Python using triple quotes'''
```

21

21

Reading input

- Syntax:

```
variable-name = input([prompt])
```

Ex: `person = input("What is your name?")`

22

22

Output using `print`

- Syntax:

  `print`(value(s), sep= ‘seperator‘, end = ‘end’)

  - `value(s) : Any value, and as many as you like. Will be converted to string before printed`
  - `Sep = 'separator' : (Optional) Specify how to separate the objects, if there is more than one. Default: ' '`
  - `end='end': (Optional) Specify what to print at the end. Default:'\n'`

23

23

Output using `print`

```
# Print data on a screen
# One object is passed
print("I am a student")

x = 5
# Two objects are passed
print("x =", x)

# code for disabling the softspace feature
print('G', 'F', 'G', sep ='/')

# using end argument
print("student", end = '@'); print("gmail.com")
```

24

24