

Lecture 2

Introduction to Python

Sequential Coding

Outline

1. Introduction
2. Variable
3. Assignment Statement
4. Data Type
5. Operators
6. Input and Output Statements
7. Sequential Coding
8. Lab Exercise

1.1 History of Python

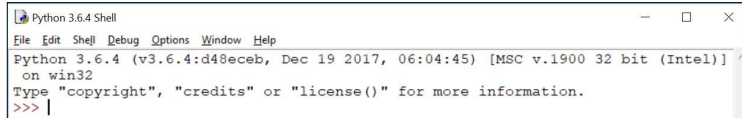
- Developed in 1989 by Guido van Rossum while working for Centrum Wiskunde & Informatica (National Research Institute for Mathematics and Computer Science) in Amsterdam, Netherlands.
- Python was first released in 1991
- Python combines features from many languages (ABC, Modula-3, C, C++, Algol-68, SmallTalk, Unix shell, etc.) with additional features such as easy-to-use interface.

1.2 The Python Language

- Python is a high-level (3GL) language, with the following characteristics:
 1. It use interpreter, translating and processing codes line-by-line, which allows:
 2. Interactive mode, allowing users to run their code one commands at a time
 3. It is currently, one of the most popular programming languages, especially for learning to program

Two Modes of Execution

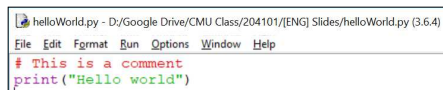
- We will use IDLE, Python's default GUI (Graphical User Interface)
- Interactive Mode
 - Work line-by-line, also serve as a main interface



```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48ebeb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> |
```

- Script Mode

- Run the whole file (a program)



```
helloWorld.py - D:/Google Drive/CMU Class/204101/[ENG] Slides/helloWorld.py (3.6.4)
File Edit Format Run Options Window Help
# This is a comment
print("Hello world")
```

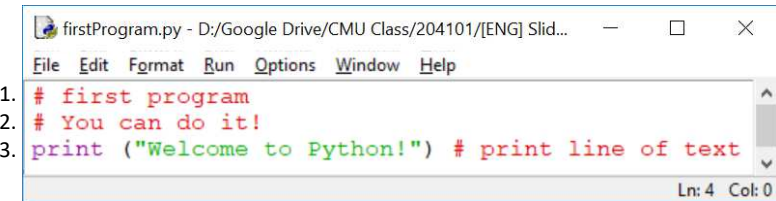
helloWorld.py



```
>>>
==== RESTART:
Hello world
>>> |
```

Output (result) on interactive mode

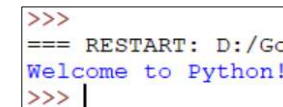
1.3 The First Program



```
firstProgram.py - D:/Google Drive/CMU Class/204101/[ENG] Slid...
File Edit Format Run Options Window Help
1. # first program
2. # You can do it!
3. print ("Welcome to Python!") # print line of text
Ln: 4 Col: 0
```

- Some Explanations

- Line 1 is a comment (start with #) comments are just notes by/for programmers, and will not be run
 - Line 2 is another comment, # will denote that the rest of the text to the end of the line is comment
 - Line 3 contains print() command, which will print string (text) inside the parentheses.
- When you run (press F5) the program, you should get the following (blue text):



```
>>>
==== RESTART: D:/Go...
Welcome to Python!
>>> |
```

2. Variable

- **Variables** are used to reserve some memory to contain data. Variables can have different **data types**, and can require different memory sizes.
- In Python, there are some rules in naming variables.
 1. It must begin with a letter (a - z, A - B) or underscore (_)
 2. Other characters can be letters, numbers (0-9) or _
 3. Cannot be reserved words
 4. No space allowed in variable's name
 5. Can be any (reasonable) length
 6. The name is case-sensitive, capital letters and lowercase letters are considered different characters.

Example: NUM, Num, and num are considered different variables

Reserved Words

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Examples of Variable Naming

Variable Name	Valid?	Why Not?
123MyID	✗	Cannot start with number
_ThinkBig	✓	
mylife@CMU	✗	Cannot contain @
Pin number	✗	Cannot contain space
Dorm_number	✓	

3. Assignment Statement

- Assignment statement is how we give value to a variable
- In other languages, variables need to be declared before usage. But in python, variables are automatically declared **the first time** they are assigned values
- We assign value to a variable using = operator, for example:

```
score = 75
```

```
# score is an integer variable with the value of 75
```

```
gpa = 2.85
```

```
# gpa is a floating point (real number) variable with the value of 2.85
```

NOTE: In Python, you can actually assign values to multiple variables in one line, for example:

```
score, gpa = 75, 2.85
```

Will do the same as the statements above.

Examples of Assignments

- You can also do chain assignment, where all the variables in the chain will be assigned the value of the rightmost variables/constant

Example 1

```
a = b = c = 1
```

```
# a, b, and c will be integers, with the value of 1
```

Example 2

```
a, b, c = 1, 2, "john"
```

```
# a and b will be integer, with values of 1 and 2 respectively
```

```
# c will be string, containing john
```

4. Data Type

- The followings are basic data types you should know:

1. `int` contains integer number
2. `float` contains real number
3. `boolean` contains True/False value
4. `string` contains text
5. `list` contains multiple items

4.1 Number Data Type (int, float)

- `int` contains integer, which are number without decimal point. Example: 2, -50, 1009
- `float` contains real number, sometime called floating point number. Real number can have decimal point. Example: 15.20, -21.9

4.2 Boolean Data Type

- Boolean variable can either be `True`, or `False`, usually a result of a comparison. For examples:
 - `4 > 1` will get you `True`
 - `6 < 7` will get you `False`
- boolean can be very useful in programming. It is use in selection (if statement) to have program does different things, based on comparison results.

4.3 String Data Type

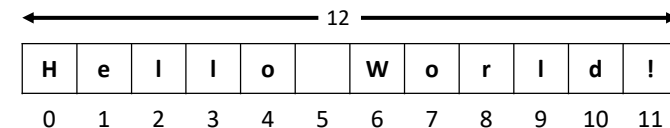
- string is used to contain text. In python string can be denote by quotation marks, and can use single (`'`), double (`"`) or triple (`'''` or `"""`) marks.
- Examples:

```
word = 'word'
sentence = "This is a sentence."
```
- Triple marks are use for string longer than a single line

```
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```
- Furthermore, you can use `+`, `*` operators
 - `+` will concatenate two or more strings together
 - `*` will repeatedly concatenate one string multiple times

4.3 String Data Type (cont.)

- You can also access part of string, which we will call `substring`, by using `[]` or `[:]` and `index` (location)
- Characters in a string are stored by index, from `0` to `end - 1`
- "Hello World!" example:



- The string has the length of 12
- The first character (index 0) is 'H'
- The last character (index 11) is "!"

Examples: Working with String

```
str = 'Hello World!'
print(str)           # Prints complete string
print(str[0])        # Prints first character of the string
print(str[2:5])       # Prints characters starting from 3rd to 5th (end - 1 is 5-1 = 4)
print(str[2:])        # Prints string starting from 3rd character
print(str * 2)        # Prints string two times
print(str + "TEST ")  # Prints concatenated string
```

This will produce the following result:

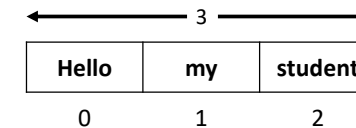
```
Hello World!
H
llo World!
Hello World!Hello World!
Hello World!TEST
```

The diagram shows a horizontal array representing the string 'Hello World!'. Each character is in its own box, with indices 0 through 11 below them. Above the array, a double-headed arrow spans from index 0 to index 11, with the number '12' centered above it, indicating the total length of the string.

H	e	l	l	o		W	o	r	l	d	!
0	1	2	3	4	5	6	7	8	9	10	11

4.4 List

- list is a container data type. A list can have multiple items.
- Items in the list can be of multiple data types (compound data type)
- When displayed, items are shown inside [] and are separated by comma (,)
- We can get values of item using [] and [:] and index values
- Like string, the first item is at index 0, and the last item is at end - 1
- Example: myList = ['Hello', 'my', 'student'] will look like this



Examples: Working with List

```
exlist1 = ['abcd', 786 , 2.23, 'john', 70.2 ]
exlist2 = ['Hello' , 'my' , 'student']

print(exlist1)        # Prints complete list
print(exlist1[0])     # Prints first element of the list
print(exlist1[1:3])   # Prints elements starting from 2nd till 3rd (end - 1 is 3 - 1 = 2)
print(exlist1[2:])    # Prints elements starting from 3rd element
print(exlist2 * 2)     # Prints list two times
print(exlist1 + exlist2) # Prints concatenated lists
```

This will produce the following result:

```
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
['Hello', 'my', 'student', 'Hello', 'my', 'student']
['abcd', 786, 2.23, 'john', 70.2, 'Hello', 'my', 'student']
```

The diagram shows a horizontal array representing the list ['abcd', 786, 2.23, 'john', 70.2]. Each element is in its own box, with indices 0 through 4 below them. Above the array, a double-headed arrow spans from index 0 to index 4, with the number '5' centered above it, indicating the total length of the list.

abcd	786	2.23	john	70.2
0	1	2	3	4

5. Operators

- Operators are symbols we used to perform some action/operations on operands (variables or constants). Today, we will look at
1. Arithmetic operators
 2. Comparison (relational) operators
 3. Assignment operators
 4. Logical operators
 5. Bitwise operators
 6. Membership operators
 7. Identity operators

5.1 Arithmetic Operators

Assume variable a holds 10 and variable b holds 20, then:

Operator	Description	Example	Result
+	Addition	a + b	30
-	Subtraction	a - b	-10
*	Multiplication	a * b	200
/	Division	b / a	2
%	Modulus	b % a	0
**	Exponent	2**3	8
//	Floor Division	9//2	4
		9.0//2.0	4.0

5.2 Comparison Operators

Assume variable a holds 10 and variable b holds 20, then:

Operator	Description	Example	result
==	equal	a == b	False
!=	not equal	a != b	True
>	greater than	a > b	False
<	less than	a < b	True
>=	greater than or equal	a >= b	False
<=	less than or equal	a <= b	True

5.3 Assignment Operators

- Beside straight assignment (=), you can also use:

Operator	Example	Explanation
=	c = a + b	
+=	c += a	c = c + a
-=	c -= a	c = c - a
*=	c *= a	c = c * a
/=	c /= a	c = c / a
%=	c %= a	c = c % a
**=	c **= a	c = c ** a
//=	c //= a	c = c // a

5.4 Logical Operators

Assume variable c holds True and variable d holds False, then:

Operator	Example	Result
and	c and c	True
	c and d	False
or	a or b	True
not	not(a or b)	False

Truth Table

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

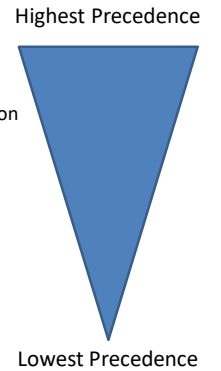
5.5 Other Operators

- There are other operators that will not be covered (yet)
 - Bitwise operators work on number bit by bit: `>>`, `<<`, `&`, `|`
 - Python Membership Operators: `in` , `not in`
 - Python Identity Operators: `is` , `is not`

5.6 Operators Precedence

- To decide which operator will be performed first, Python follow the precedence denote in the table below:
- For operators with the same precedence, perform the left one first

Operator	Description
()	parentheses
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^	Bitwise exclusive 'OR' and regular 'OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //=- = += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators



Translating Mathematical Expression into Python

- You might be familiar with a certain style of writing an expression, but it might not be understood by Python, and you need to convert them.
- Examples:

Mathematical Expression	Python Expression
$xy - 5z$	<code>x * y - 5 * z</code>
$x^2 + 4y + 5$	<code>x**2 + 4 * y + 5</code>
$8y^2 - 8z$	<code>8 * y**2 - 8 * z</code>

Example: Operators Precedence #1

Find the result of $c * d - x / y$, with the following assignments

`x = 16`

`y = 4`

`c = 2.5`

`d = 0.25`

Solution

`c * d - x / y`

`2.5 * 0.25 - 16 / 4` # assignment

`0.625 - 16 / 4` # perform *

`0.625 - 4` # perform /

`-3.375` # perform -

The answer is -3.375

Example: Operators Precedence #2

- Find the result of $-(-5) * (x \% y - x // (y+1))$, with the following assignments

$x = 16$
 $y = 4$
 $c = 2.5$
 $d = 0.25$

Solution

```

$$\begin{aligned} &-(-5) * (x \% y - x // (y+1)) \\ &-(-5) * (16 \% 4 - 16 // (4+1)) \\ &-(-5) * (16 \% 4 - 16 // 5) \\ &5 * (16 \% 4 - 16 // 5) \\ &5 * (0 - 3) \\ &5 * (-3) \\ &-15 \end{aligned}$$

```

The answer is -15

Example: Operators Precedence #3

- Find the result of $c // d + y \% x != 2*c - d$, with the following assignments

$x = 16$
 $y = 4$
 $c = 2.5$
 $d = 0.25$

Solution

```

$$\begin{aligned} &c // d + y \% x != 2*c - d \\ &2.5 // 0.25 + 4 \% 16 != 2*2.5 - 0.25 \\ &10 + 4 != 5 - 0.25 \\ &14 != 4.75 \end{aligned}$$

```

True

The answer is True

Some Useful Functions in Python

• int(A)

- Convert argument (Value of variable A, in this case) into integer
- Also need a variable to assign the integer to
- Try $A = \text{int}(3.142)$ → What is the value of A?

• float(A)

- Similar to int(), but will convert the argument into floating point number instead

Some Useful Functions in Python (cont.)

- ❖ Note: you can have nested function call (one function inside another). The innermost function will be called first, and the result will be argument for the one right outside it
- ❖ For example: $A = \text{int}(\text{input}(\text{"Enter a number"}))$
 - ❖ $\text{input}(\text{"Enter a number"})$ will be called first, prompting the user to enter the number
 - ❖ The result of $\text{input}()$, which is a string, will then be converted into an integer by $\text{int}()$, and then assigned to variable A

6. Input-output Statements (Recap)

- For a program to be able to interact with the user, the program need a way to:
 - Receive data from the user (input), and
 - Display result to user (output)
- We will start with:
 - Output: print() function
 - Input: input() function

6.1 print()

- print() will display its arguments (values inside the parentheses) on interactive mode windows.
- You can print multiple strings/variables at the same time by separating them with comma (.). The texts will be concatenated when printed.
- If you want the text to start a new line, you can add new line character (\n) in the text
- You can also use \t for tab

6.2 input()

- input() function will prompt user (with the message inside the parentheses) to enter data. User will type out data, then press enter.
- Received input will be of string type, and we need a variable to hold the data (function return)

Example:

```
str = input("Enter your input: ")  
print ("Received input is : ", str)
```

Result (user input in blue):

```
Enter your input: Hello Python  
Received input is : Hello Python
```

- From the example, "Hello Python" will be stored in variable `str`

6.2 input() (cont.)

- Since the input will be string (text), it cannot be used in calculation right away
- We will need to convert the input into appropriate type first (int or float)

Example:

```
inp1 = input("Input integer number : ") # prompt user for input  
                                         # store the result on inp1  
no1 = int(inp1)                         # convert result in inp1 to int  
                                         # store it on no1  
inp2 = input("Input float number : ")  # get next user input into inp2  
no2 = float(inp2)                      # convert to float, store on no2
```

6.2 input() (cont.)

- Warning: be careful about input data type.

Example:

```
inp=input("Input number : ")
```

```
no=int(inp)
```

```
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    no = int(inp)
ValueError: invalid literal for int() with base 10: '1.2'
```

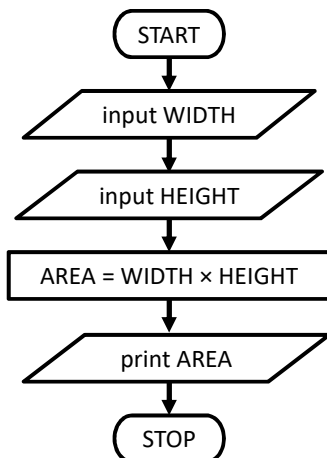
- If the user input 1.2, it will read and put into inp with problem. BUT when Python try to convert it to integer, an error will occur (since 1.2 is not an integer)

7. Writing Sequential Code

- Translate from design (description, pseudocode, or flowchart) into programming code
- You might need to do program analysis and program design first!
- Start from the top, moving downward and end at the bottom

7.1 Coding Example #01

- Finding rectangle's area



```
#Receive length, width from input
LENGTH = float(input("Input length : "))
WIDTH = float(input("Input width : "))

#Calculate area
#using formula "area = length * width"
AREA = LENGTH * WIDTH

#Display the result
print("Area =", AREA)
```

7.2 Coding Example #02

- Swap the values of two variables

“Take two already-assigned variables, A and B. Swap the values between them, then print the result”

```
#Start with assignment
A = 5
B = 7
print("A =", A)
print("B =", B)

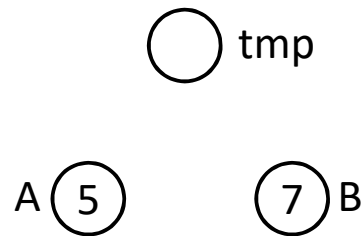
#Put code for swapping A and B here

#End swapping

#Show the results
print("A =", A)
print("B =", B)
```

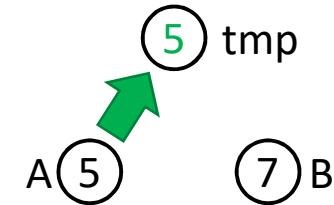
7.2 Coding Example #02 – Swapping Concept

- VERY IMPORTANT: A code will be executed one line at a time, so If you do `A = B` first, the original value of A will be lost
- You will need temporary variable (tmp)

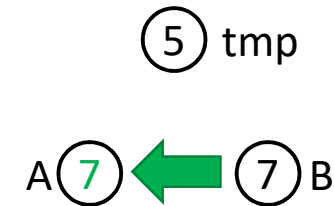


7.2 Coding Example #02 – Step by Step

1. `tmp = A` # assign the value of A to tmp

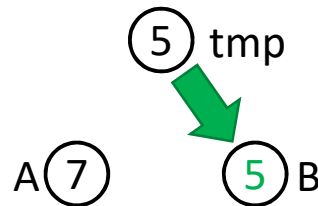


2. `A = B` # assign the value of B to A



7.2 Coding Example #02 – Step by Step (cont.)

3. `B = tmp` # assign value of tmp (A's original values) to B



- And we have `A = 7` and `B = 5`!

7.2 Coding Example #02 – Complete Codes

```
#Start with assignment
A = 5
B = 7
print("A =", A)
print("B =", B)

#Put code for swapping A and B here
temp = A # use temp as temporary to hold A's value
A = B    # assign B's values to A
B = temp # assign temp's (A's original) values to B
#End swapping

#Show the results
print("A =", A)
print("B =", B)
```

7.3 Coding Example #03

- Finding Mean Average of Three Numbers

“Receive three numbers from the input, then display the mean of those three numbers”

- Note: use `float()` for the inputs
- The program should work like this:

```
Input first number : 14.32
Input second number : 15.44
Input third number : 17.29
Mean = 15.683333333333332
>>> |
```

7.3 Coding Example #03 – General Idea

❖ So first, some questions:

- What is(are) the input(s) of this program?
- Output(s)?
- Process? Or, how do we calculate the mean of three numbers?

❖ Can you then describe the algorithm?

- What are the steps you need to do to solve the problem, starting from: where are you going to get the input data from?

- If you're not sure about the codes yet, write the steps down in comments first, then do the coding for each step

```
# received three number from input
num_01 = float(input("Input first number : "))
num_02 = float(input("Input second number : "))
num_03 = float(input("Input third number : "))

# calculate the mean, which is the sum of the numbers
# divided by the count (3)
meanNum = (num_01 + num_02 + num_03) / 3

# show the mean
print("Mean =", meanNum)
```

Example #03 – Complete Codes

```
# calculate the mean, which is the sum of the numbers
# divided by the count (3)
meanNum = (num_01 + num_02 + num_03) / 3

# show the mean
print("Mean =", meanNum)
```

Practice

- Find BMI
- Find an average of three numbers
- Find the circumference of a circle
- Find the area of a circle

Reference

- Mark Lutz, “Programming Python”, O'Reilly, 1996.
- Deitel, “Python How to program”, Prentice-Hall, Inc., 2002.
- Matt Telles, “Python Power!”, Thomson Course Technology, 2008.
- python 3 help documentation
- PYTHON TUTORIAL Simply Easy Learning by tutorialspoint.com