Introduction to Python

NEE2106 Computer Programming for Electrical Engineers

Prepared by: Dr. Rui Li (rui.li@vu.edu.au)

VICTORIA UNIVERSITY
MELBOURNE AUSTRALIA

< Contents >

< Schedule and Assessment >

Session	2hr Workshop	2hr Computer Lab	Extra 1hr	Assessments
S1	Introduction to Python Python basics	Computer Lab 1		Lab exercise (5%)
S2	Iteration Conditional statements	Computer Lab 2		Lab exercise (5%)
S3	Data I/O	Computer Lab 3		
S4	Python libraries	Computer Lab 4	Test 1 (20%)	Lab exercise (5%)
S5	Python for statistics	Computer Lab 5		Lab exercise (5%)
S6	Simple GUI	Computer Lab 6		
S7	Introduction to PBL Project		Test 2 (20%)	
S8	GUI design for PBL: 1) data visualisation; 2) user controls			Project part 1 submission (10%)
S9	Introduction to Machine learning		Milestone demonstration	
S10	Apply ML features in PBL and finalise the project			
S11			Poster presentation (10%)	Project part 2 submission (20%)

< Contents >

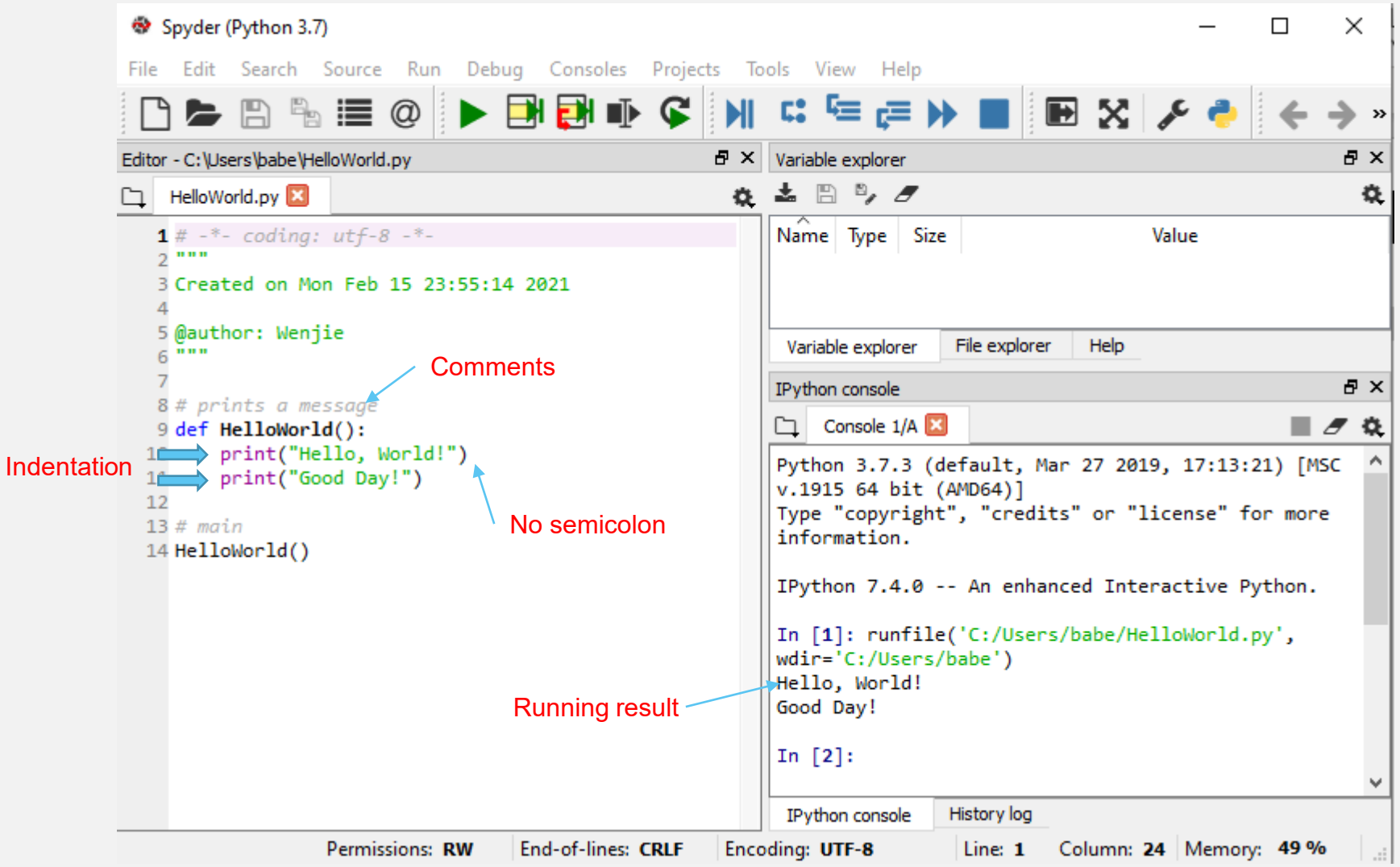
< Python is >

- A scripting language
- An interpreted, high-level and general-purpose programming language
- Widely used in many cutting-edge technologies (AI, IoT, etc.)
- Well known for its simplicity, readability, and community support (developers contributed various libraries, frameworks, and tools)

< Application >

- Scientific and Numeric Applications
- Desktop GUI
- Artificial Intelligence and Machine Learning
- Enterprise-level/Business Applications
- Web development
- Game development

< Introduction to Python >



< Python Environment – Anaconda IDE >

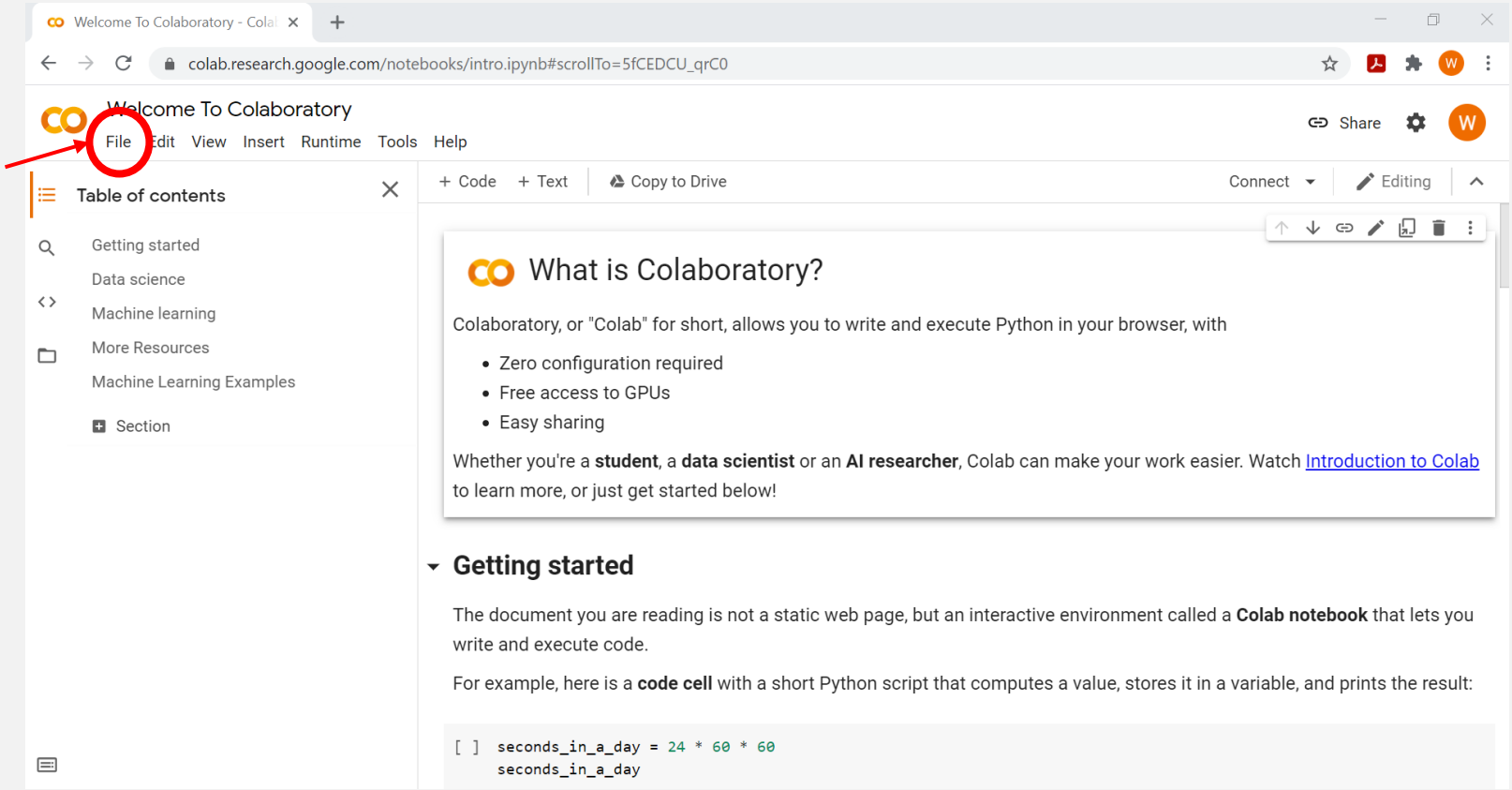
1. Download Anaconda with Python 3.X: https://www.anaconda.com/products/individual

Windows	MacOS	Linux
Python 3.8	Python 3.8	Python 3.8
64-Bit Graphical Installer (457 MB)	64-Bit Graphical Installer (435 MB)	64-Bit (x86) Installer (529 MB)
32-Bit Graphical Installer (403 MB)	64-Bit Command Line Installer (428 MB)	64-Bit (Power8 and Power9) Installer (279 MB)

2. Choose a suitable installer for your PC and run it
3. Follow the installation instructions
4. Run Spyder editor and create "HelloWorld.py"

< Python Environment – Google Colab>

1. You need a Google account
2. Create a new Notebook in Google Colab: https://colab.research.google.com/notebooks/intro.ipynb



< Example >

Add code pieces

Untitled2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[3] # define a function
    def hi():
        print('Hello World!')  # contents of the function
        print('This is RUI ~')
    # call function
    hi()
```

```
Hello World!
This is RUI ~
```

```
# print the msg directly
print('Hello World!')
print('This is RUI ~')
```

```
Hello World!
This is RUI ~
```

Code

Results

Code

Results

< Contents >

< Variable >

- A variable is a reserved memory location to store value
- A variable can have a short name (i.e. x, y) or a more descriptive name (i.e. name, height, index)

- Naming rules:
  - must start with a letter or the underscore character
  - cannot start with a digit
  - only contain alpha-numeric characters and underscores (A-Z, a-z, 0-9, and _)
  - case sensitive (Name, name and NAME are different variables)

- Symbolic constants: variables that contain values that never changes in the program
  - named by using all uppercase letters
  - e.g.: TAX_RATE and STANDARD_DEDUCTION

< Example >

Identify legal/ illegal variable names.

Legal names	Illegal names
name total_amount isValid _height x1 y12 year_	na me total-amount 2people -height x1% y12$ year-

Name
na me
total-amount
total_amount
isValid
2isValid
-height
_height
x1
X1%
Y$12
y12
*year_*
Year-

< Data Type >

Consider this sentence:

"In 2007, Michelle paid $120,000 for her house at 24 East Maple Street with a loan rate of 3.45%."

What are the data variables in this sentence?

a date (2007), → int
a name (Micaela), → str
a price ($120,000), → int
an address (24 East Maple Street) → str
a loan rate (3.45%) → float

- **A data type** consists of a set of values and a set of operations that can be performed on those values.
- **A literal** is the way a value of a data type looks to a programmer.

TYPE OF DATA	PYTHON TYPE NAME	EXAMPLE LITERALS
Integers	`int`	`-1, 0, 1, 2`
Real numbers	`float`	`-0.55, .3333, 3.14, 6.0`
Character strings	`str`	`"Hi", "", 'A', '66'`

< String >

- In programming, text is referred to as a string, enclosed in quotation mark "….." , or '…..'
- The **print** function displays a string without the quotation marks

s1.py*

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Apr  5 11:36:57 2024

@author: e5107499
"""

# define the string
'HELLO WORLD!'

# define a variable that contains a straing
a = "Hi there "

# combine 2 strings
name = "Rui"
a+name

# display the results
print(a+name)
print("Hello World, it's ", name)
```

Name	Type	Size	
a	str	9	Hi there
name	str	3	Rui

```
In [1]: runfile('C:/Users/e5107499/Desktop/NEE210
NEE2106/2024')
Hi there Rui
Hello World, it's  Rui
```

< Escape String >

ESCAPE SEQUENCE	MEANING
\b	Backspace
\n	Newline
\t	Horizontal tab
\\	The \ character
\'	Single quotation mark
\"	Double quotation mark

```
#%% multiple lines and a tab
print(" \"This is Rui \n \t from \b Vic Uni\" ")
```

```
"This is Rui
     from Vic Uni"
```

Use three consecutive quotation marks (either single or double) for multi line printing.

```
#%% multiline
print(""" I have a very long sentence
 that I want to display
 on 3 lines""")
```

```
I have a very long sentence
that I want to display
on 3 lines
```

< Joint String >

Join two or more strings to form a new string:

```
#%% join 3 strings
print("Hi" + " there" + " Rui")
```

```
Hi there Rui
```

The * operator allows to build a string by repeating another string a given number of times:

```
#%% repeat string
print("Hi! "*10 + "Rui")
```

```
Hi! Hi! Hi! Hi! Hi! Hi! Hi! Hi! Hi! Hi! Rui
```

< Example >

Print multiple output on the same line using **end=‘ ’**

Enter this code in Python IDE then click **run**

Examples2.py -

File Edit Format Run Options

```
print('A', end='')
print('B', end='')
print('C', end='')
print()
print('X')
print('Y')
print('Z')
```

```
ABC
X
Y
Z
>>>
```

< Example >

Ask for user input using **input** function

```
#%% user input
name=input("Enter your name: ")
print("hello",name)
```

```
In [9]: runcell(2, 'C:/Users/e5107499/Desktop/NEE2106/2024/s1.py')
Enter your name: Rui
hello Rui
```

Try this code:

number = input("please enter a number: ")

number + 5

What do you get?

input function always builds a string from the user's keystrokes.

Conversion function is needed to convert String to Number:

- int() – convert to integers
- float() – convert to floating-point numbers

```
#%% input a number
num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))
print("The sum is ",num1+num2)
```

```
Enter the first number: 2
Enter the second number: 5
The sum is  7
```

< Numeric Data Type>

- Integer data type: **int** consists of the integers from 2,147,483,648 (-2^{31}) to 2,147,483,647 ($2^{31} - 1$)
- Floating-point number: **float** represents real numbers range from approximately -10^{308} to 10^{308} and have 16 digits of precision
- Complex number: written with a "***j***" as the imaginary part, for example *3 + 5j*
- **Type conversion** function is needed when working with the input of numbers

CONVERSION FUNCTION	EXAMPLE USE	VALUE RETURNED
int(<a number or a string>)	int(3.77)	3
	int("33")	33
float(<a number or a string>)	float(22)	22.0
str(<any value>)	str(99)	'99'

< Formatting Control >

Right-align or left-align the string “four” within a specified field width of 6.

% : Format Operator

```
>>> "%6s" % "four"          # Right justify
'  four'
>>> "%-6s" % "four"         # Left justify
'four  '
```

Positive field width – Right Justified
Negative field width – Left justified

s – displayed data for string
(use **d** for integer, **f** for floating points)

6: Field width

Output of a floating-point number without format string, with precision, with field width and precision.

```
>>> salary = 100.00
>>> print("Your salary is $" + str(salary))
Your salary is $100.0
>>> print("Your salary is $%0.2f" % salary )
Your salary is $100.00
>>> print("Your salary is $%10.2f" % salary )
Your salary is $    100.00
>>>
```

%<length>.<percision><type>

d	int
f	float
s	string

< Example >

Write a program to calculate the following:

A dozen eggs cost $2.39.
Display on screen the egg price for the end user.
Ask the user how many dozens he wants to buy and how much cash he will pay.
Calculate and display the change.

```
Today's egg price is $ 2.39 per dozen
How many dozens do you want? 2
How much cash will you give me? 10.5
Your change is $ 5.72
```

```python
#%% sell eggs
price = 2.39 # price per dozen
print("Today's egg price is $",price,"per dozen")
num = int(input("How many dozens do you want? "))
pay = float(input("How much cash will you give me? "))
print("Your change is $",pay-num*2.39)
```

< Contents >

< Arithmetic Expression >

OPERATOR	MEANING	SYNTAX
-	Negation	-a
**	Exponentiation	a ** b
*	Multiplication	a * b
/	Division	a / b
//	Quotient	a // b
%	Remainder or modulus	a % b
+	Addition	a + b
-	Subtraction	a - b

e.g. Quotient vs. Division:

`3 // 2 * 5.0` yields `1 * 5.0`, which yields `5.0`

whereas

`3 / 2 * 5` yields `1.5 * 5`, which yields `7.5`

< Arithmetic Expression >

The **round()** function rounds a float to the nearest int

```
>>> int(6.75)
6
>>> round(6.75)
7
```

The **round(val,n)** function rounds a float to n decimals

```
In [6]: round(6.123456,2)
Out[6]: 6.12
```

Move to new line by backslash character **** at the end of the current line.

```
>>> 3 + 4 * \
2 ** 5
131
>>>
```

< Example >

Let **x** = 8 and **y** = 2. Write the values of the following expressions:

a	`x + y * 3`	14
b	`(x + y) * 3`	30
c	`x ** y`	64
d	`x % y`	0
e	`x / 12.0`	0.6666
f	`x // 6`	1

< Contents >

< Functions >

Function is a chunk of code that can be called by name to perform a task.

Arguments are specific data values required for function to perform its task.

Returning value(s) is(are) the output value of a function.

function name

argument

```
>>> round(5.6)
6
```

returning value

Values returned by function calls can be used in expressions and statements

```
>>> print(abs(4 - 5) + 3)
```

< Modules >

Modules are components where functions and other resources are coded.

- Built-in module: always available for use directly.
- Other modules: need to be imported.

e.g. **math** module includes several functions that perform mathematical calculations

```
>>> import math
>>> dir(math)
['__doc__', '__file__', '__name__', '__package__', 'acos', 'acosh', 'asin',
'asinh', 'atan', 'atanh', 'ceil','copysign', 'cos', 'cosh', 'degrees'
'exp', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'hypot'
'isinf', 'isnan', 'ldexp', 'log', 'log10', 'log1p', 'modf', 'pi', 'po
'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

Import a module

Show all the functions in this module

```
>>> math.pi
3.141592653589793
```

Use source from the **math** module

```
>>> from math import pi, sqrt
>>> sqrt(4)
2.0
>>> pi
3.141592653589793
```

Import the individual resources

```
In [15]: help(math)
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module provides access to the mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

        The result is between 0 and pi.
```

Browse the entire documentation

< Example >

Write a Python program that calculates the area and circumference of a circle given its radius. Use the **math** module to perform the necessary calculations.

```
the radius is: 3.1
the area is 30.19  and circumference is  19.48
```

```
#%% area and circumference of a circle
r = float(input('the radius is: '))
from math import pi
area = round(pi*r**2,2)
cir = round(pi*r*2,2)
print("the area is",area," and circumference is ",cir)
```