

# PYTHON: EXPERIENCING IDLE AND WRITING SIMPLE PROGRAMS

---

# About Me

- Subject lecturer: Kevin K.F. Yuen (Ph.D.)
- Office: PQ732
- E-Mail: [kevin.yuen@polyu.edu.hk](mailto:kevin.yuen@polyu.edu.hk)
- Consultation hour: flexible arrangement by email appointment.

# General Information about this course

- Hybrid teaching
- Room: PQ604
- Online Interactions in MS teams
- Lecture and Lab Notes: <https://learn.polyu.edu.hk>

# About coding for beginners

不聞不若聞之，  
聞之不若見之，  
見之不若知之，  
知之不若行之。  
學至於行之而止矣。  
**行**之，**明**也。  
-- 《荀子·儒效》

I hear and I forget.  
I see and I remember.  
I **do** and I **understand**.

“When we are in the bottom of the mountain, we think the mountain is so high. However, don’t scare.  
When we reach the top of mountain, we will find the ground is so close to us!  
Next we will challenge the higher mountain. Don’t hesitate! Just go ahead!”

# Why Python

What is the rank of Python in 2021 or 2022?

Language Ranking: IEEE Spectrum

Rank	Language	Type	Score
1	Python		100.0
2	Java		95.3
3	C		94.6
4	C++		87.0
5	JavaScript		79.5
6	R		78.6
7	Arduino		73.2
8	Go		73.1
9	Swift		70.5
10	Matlab		68.4

**2020**

Language Ranking: IEEE Spectrum

Rank	Language	Type	Score
1	Python		100.0
2	Java		96.3
3	C		94.4
4	C++		87.5
5	R		81.5
6	JavaScript		79.4
7	C#		74.5
8	Matlab		70.6
9	Swift		69.1
10	Go		68.0

**2019**

Language Rank Types Spectrum Ranking

1.	Python		100.0
2.	C++		99.7
3.	Java		97.5
4.	C		96.7
5.	C#		89.4
6.	PHP		84.9
7.	R		82.9
8.	JavaScript		82.6
9.	Go		76.4
10.	Assembly		74.1

**2018**

Language Rank Types Spectrum Ranking

1.	Python		100.0
2.	C		99.7
3.	Java		99.5
4.	C++		97.1
5.	C#		87.7
6.	R		87.7
7.	JavaScript		85.6
8.	PHP		81.2
9.	Go		75.1
10.	Swift		73.7

**2017**

Language Rank Types Spectrum Ranking

1.	C		100.0
2.	Java		98.1
3.	Python		98.0
4.	C++		95.9
5.	R		87.9
6.	C#		86.7
7.	PHP		82.8
8.	JavaScript		82.2
9.	Ruby		74.5
10.	Go		71.9

**2016**

Source: [spectrum.ieee.org/static/interactive-the-top-programming-languages-2020](https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2020)

## Ask, Don't Stare

If you write code, you will write bugs. A "bug" means a defect, error, or problem with the code you've written. The legends say that this comes from an actual moth that flew into one of the first computers causing it to malfunction. Fixing it required "de-bugging" the computer. In the world of software, there are a *lot* of bugs. So many.

Like that first moth, your bugs will be hidden somewhere in the code, and you have to go find them. You can't just sit at your computer screen staring at the words you've written hoping that the answer jumps out at you. There is no more additional information you can get doing that, and you need additional information. You need to get up and go find the moth.

To do that you have to interrogate your code and ask it what is going on or look at the problem from a different view. In this book I'll frequently tell you to "stop staring and ask". I'll show you how to make your code tell you everything it can about what's going on and how to turn this into possible solutions. I'll also give you ways to see your code in different ways, so you can get more information and insight.

## Do Not Copy-Paste

You must *type* each of these exercises in, manually. If you copy and paste, you might as well not even do them. The point of these exercises is to train your hands, your brain, and your mind in how to read, write, and see code. If you copy-paste, you are cheating yourself out of the effectiveness of the lessons.

- No source codes cleaned will be provided for this course.
- You must type and code by yourself.
- The classes are not to aim to provide solutions, but expect learners to learn from mistakes.

There are vast of source codes in the internet. You may try them in the future.

# Optional readings from my previous classes

- <https://github.com/kkfyuen/PythonWorkshop2020>
- <https://github.com/kkfyuen/ANL251Python>



# Objectives

- To be able to use Integrated Development and Learning Environment (IDLE) to program in Python
- To be able to understand and write Python statements to
  - output information to the screen
  - assign values to variables
  - get numeric information entered from the keyboard

# Python

- Created by Guido van Rossum, National Research Institute for Mathematics and Computer Science in the Netherlands in the late 1980s
- A scripting language based on ABC
- Supports structured programming and object-oriented programming
  - Easy to extend
- The latest version is 3.8.5



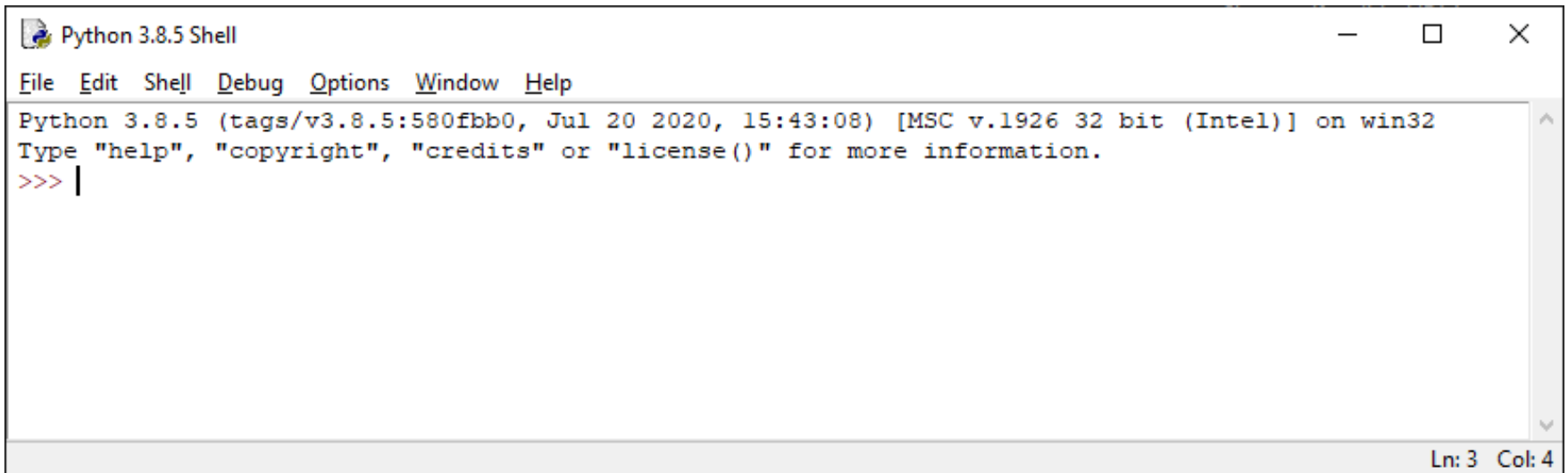
# Integrated Development and Learning Environment (IDLE)

- Support both Windows and Mac OS



# IDLE

- An interactive environment (called “Shell”)

A screenshot of the Python 3.8.5 Shell window. The window has a title bar that says "Python 3.8.5 Shell" and standard Windows window controls (minimize, maximize, close). Below the title bar is a menu bar with the following items: File, Edit, Shell, Debug, Options, Window, and Help. The main area of the window contains the following text: "Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32", "Type 'help', 'copyright', 'credits' or 'license()' for more information.", and a prompt ">>> |". The status bar at the bottom right of the window shows "Ln: 3 Col: 4".

```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> |
Ln: 3 Col: 4
```

# "Hello World!"

- Type the followings:

```
print("Hello World!")  
print("Hello World!\n")  
print("Hello\nWorld!\n")
```

- What do you see?

# EXERCISE 1

---

Display your name and student ID on the screen. Skip one line before printing the student ID

# Input and Output

- Type the followings:

```
print("Hello World!")  
name = input("What is your name? ")  
Dennis  
print(name)
```

- What do you see?

# Input, Process, Output (IPO)

- Type the followings:

```
x = eval(input("Input an integer: "))  
#123  
y = eval(input("Input another integer: "))  
#1001  
sum = x + y  
print(sum)
```

- What do you see?



# Example Program:

## Temperature Converter

- Problem: The temperature is given in Celsius, user wants it expressed in degrees Fahrenheit
- Specification
  - Input – temperature in Celsius
  - Output – temperature in Fahrenheit
    - $\text{Output} = 9/5 \times (\text{input}) + 32$

# Example Program: Temperature Converter

- Design
  - Input, Process, Output (IPO)
    - Prompt the user for input (Celsius temperature)
    - Process it to convert it to Fahrenheit using  $F = 9/5 \times (C) + 32$
    - Output the result by displaying it on the screen

# Example Program: Temperature Converter

- Before we start coding, let's write a rough draft of the program in *pseudocode*
- Pseudocode is precise English that describes what a program does, step by step
- Using pseudocode, we can concentrate on the algorithm rather than the programming language

# Example Program: Temperature Converter

- Pseudocode:
  - I - Input the temperature in degrees Celsius (call it celsius)
  - P - Calculate fahrenheit as  $(9/5) \times \text{celsius} + 32$
  - O - Output fahrenheit
- Now we need to convert this to Python!
- Besides entering each statement in IDLE, we can create a source file with an extension `.py` that contains the Python code

# Example Program: Temperature Converter

```
# convert.py
# A program to convert Celsius temps to Fahrenheit
# by: Susan Computewell

def main():
    celsius = eval(input("What is the Celsius temperature? "))
    fahrenheit = 9/5 * celsius + 32
    print("The temperature is", fahrenheit, "degrees Fahrenheit.")

main()
```

# Saving the program in a .py file.

- Programs are usually composed of functions, *modules*, or *scripts* that are saved on disk so that they can be used again and again
- A *module file* is a text file created in text editing software (saved as “plain text”) that contains function definitions
- In our example, we’ll use *convert.py* when we save our work to indicate it’s a Python program
- Remember that Python treats indentation seriously: each indentation in a Python program must strictly consist of **4 spaces**, no more and no less

# Getting \*.py to Run

- Under **File** in IDLE, select **Open** to open the source file
- Under the open file window, select **Run** -> **Run Module** or Press **F5** on the keyboard
- Do not forget to include the last line **main()** in your Python program
  - The detailed role of **main()** will be discussed later

# Example Program: Temperature Converter

- Here is the program behaviour

```
>>>
```

```
What is the Celsius temperature? 0
```

```
The temperature is 32.0 degrees Fahrenheit.
```

```
>>> main()
```

```
What is the Celsius temperature? 100
```

```
The temperature is 212.0 degrees Fahrenheit.
```

```
>>> main()
```

```
What is the Celsius temperature? -40
```

```
The temperature is -40.0 degrees Fahrenheit.
```

```
>>>
```



# EXERCISE 2

Modify `convert.py` so that it will take the degree in Fahrenheit and change it back to Celsius

---

How can we modify the output below?

```
>>>
What is the Celsius temperature? 100
The temperature is 212.0 degrees Fahrenheit.
The temperature is 100.0 degrees Celsius.
>>>
```

# Naming the identifiers

- Names are given to variables (`celsius`, `fahrenheit`), modules (`convert`), functions (`main`), etc
- These names are called *identifiers*
- Every identifier must begin with a letter or underscore (“\_”), followed by any sequence of letters, digits, or underscores
  - Note that “-”, “.” and many other symbols are not allowed
- Identifiers are case sensitive
- Identifiers cannot be Python’s keywords

# Python's keywords

Keywords in Python programming language

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

# Expressions

- The fragments of code that produce or calculate new data values are called *expressions*
- A (numeric/string) *literal*, which is the simplest kind of expression, is used to represent a specific value, e.g., 10 or "Dennis"
  - A simple identifier can also be an expression.
- Simpler expressions can be combined using *operators* +, -, \*, /, //, \*\*, %
  - The normal mathematical precedence applies.
  - Only round parentheses can be used to change the precedence, e.g., ((x1 - x2) / 2\*n) + (spam / k\*\*3)
- Try "I" + "love" + "you"

# EXERCISE 3

---

If you assign 5 to `x` and then type `x`, the shell will return 5 to you. What if you type `y` but without assigning any value to it before?

# Assignment statements

- Simple assignment: `<variable> = <expr>`

`<variable>` is an identifier, `<expr>` is an expression

- The expression on the RHS is evaluated to produce a value which is then associated with the variable named on the LHS

# Assigning Input

- The purpose of an input statement is to get input from the user and store it into a variable.

```
<variable> = input(<prompt>)
```

- E.g.,

```
x = eval(input("Enter a temperature in Celsius: "))
```

- Print the prompt
- Wait for the user to enter a value and press <enter>
- The expression that was entered is evaluated and assigned to the input variable
- Two kinds of input: character string or number

# EXERCISE 4

---

- Prompt user for a number and then print it out using just `input(<prompt>)`
- Prompt user for a number and then print it out using `eval(input(<prompt>))`
- What is the difference between the two?
- Remove `eval()` from `convert.py` and does it still run? Why?



# Summary

- The Integrated Development and Learning Environment (IDLE) to program in Python is introduced
- We have examined Python statements to
  - output information to the screen
  - assign values to variables
  - get numeric information entered from the keyboard

END

---