Learn Python with PyGame

Part 1
Introduction to Python

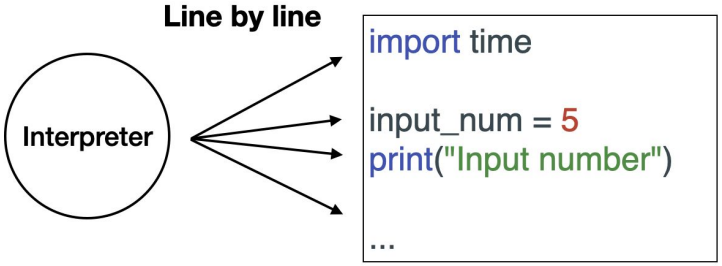
What is Python?

Python is an **interpreted, object-oriented, high-level programming language** with dynamic semantics.

(Fig.) Logo of Python

Interpreter

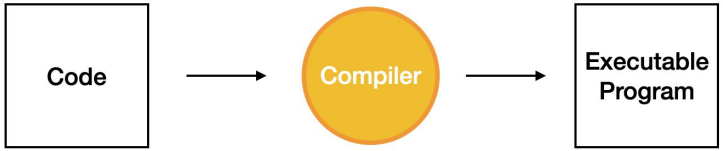
The interpreter will run through a program **line by line** and execute each command in a line.



(Fig.) Interpreter

Compiler

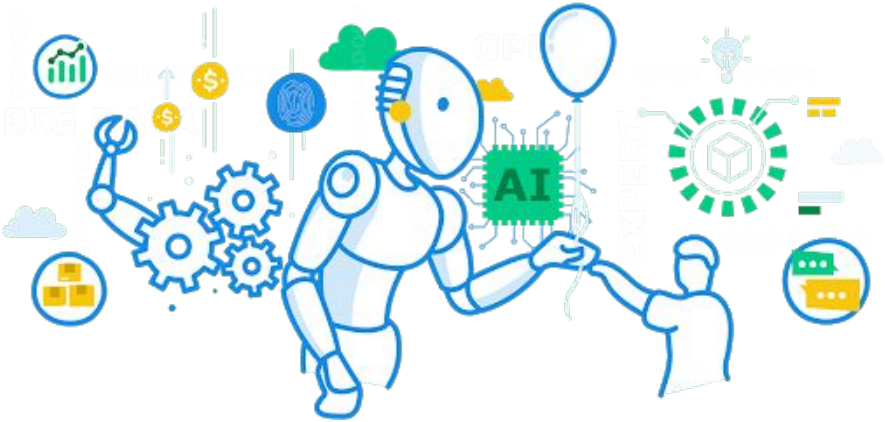
After finishing compile the code, the complier will generate the **executable program**.



(Fig.) Compiler

Part 2
Application

Artificial Intelligence



(Fig.) AI illustration

Web Server

(Fig.) Frameworks for webserver

Robotic operating system

(Fig.) ROS illustration

Part 3
Python Syntax

Hello World!

- To print the string, we use print(). For example, let's print "Hello World!".

Code

```
print("Hello World!")
```

Exercise 1

- Print out your name!

Answer 1

Code

```
print("<your name>")
```

Comment (1 / 2)

- Comment can **explain the code** to the readers (include owners) and also make the code **more readable**.
- To comment a single line, use `#`.
- To comment multiple lines, use `'''` to quote the wanted lines.

Comment (2 / 2)

Code

```
# print out my name

print("Justin")
```

Code

```
'''
Print
Hello
World
'''
print("Hello world")
```

Types (1 / 4)

- Types are simply the categories of all the objects.
- Python is dynamical type. Therefore, the type of variables are decide in runtime.

Dynamically

```python
# In python
number_int = 2
number_int = 2.0
number_int = "2"
```

Statically

```cpp
// In c++
int number_int = 2;
float number_float = 2.0;
const char* number_str = "2";
```

(Fig.) Different types of language

Types (2 / 4) - Number

- Python mainly support two types of number - Integer and Float.
- The built-in function type() can help us figure out the type.

Code

```
# Output the type of the number 7
print(type(7))

# Output the type of the number 7.5
print(type(7.5))
```

Types (3 / 4) - String

- String is defined either with the single quote or double quote.
- Remind that 7 is **not the same** as "7" or '7'.

Code

```
# Output the type of the string
print(type("Hello"))
print(type('Hello'))
```

Types (4 / 4) - Bool

- Boolean type include only two instances - True and False.

Code

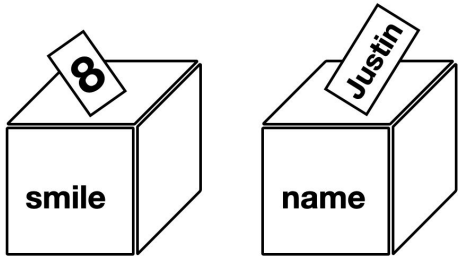
```
# Output the type of True and False
print(type(True))
print(type(False))
```

Variables (1 / 2)

- We can think of variable as a **box**. Each box has a **name** and **value**.
- To assign value to the variabel, we simply use `=` to connect the name and value.

Code

```
# Assign number 8 to variable named smile
smile = 8

# Assign string "Justin" to variable
#  named name
name = "Justin"
```



(Fig.) Boxes with name and value

Variables (2 / 2)

- Everytime we refer the variable's name, we can get it value.
- To check this, we can simply use print().

Code

```
# Print out the value of the variables
print(smile)
print(name)
```

Exercise - 2

- Assign "Here you go" to the variable named phrase and print out its type

Answer - 2

Code

```
# Assign value to phrase
phrase = "Here you go"

# Print out its type
print(type(phrase))
```

Operators (1 / 5) - Arithmetic

- Arithmetic operators can used to do the common mathmatical operation

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Operators (2 / 5) - Comparison

- Comparison operators can be used to evaluate the relationship between two things. Return type is **boolean**.

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Smaller than	x < y
>=	Greater than or equal to	x >= y
<=	Smaller than or equal to	x <= y

Operators (3 / 5) - Logical

- Logical operators are used to combine the comparison or conditional statements.

Operator	Description	Example
and	Return true if both are ture	x and y
or	Return true if either is true	x or y
not	Return the opposite of the statement	not x

Operators (4 / 5) - Truth table

- Truth table can shown the result of different logical combination.

x	y	(x and y)
F	F	F
F	T	F
T	F	F
T	T	T

x	y	(x or y)
F	F	F
F	T	T
T	F	T
T	T	T

x	not x
F	T
T	F

(Fig.) Truth table generated from here

Operators (5 / 5)

Code

```
# Addition
print(2 + 3)

# Subtraction
print(2.5 - 3.1)

# Multiplication
print(2.5 * 4)
```

Code

```
# Equal
print(2 == 3)

# Greater than
print(6 > 7)

# Greater than or equal to
print(5 >= 3)
```

Code

```
# True and True
print(True and True)

# True or False
print(True or False)

# Not False
print(not False)
```

Exercise - 3

- Please fill in the value of x, y and c.

Code

```
# x = (True and False) or False
x = # True of false

# y = (2 + 3) ** 2
y = # Numeric value

# c = (a > b)
a = 9
b = 6
c = # True or False
```

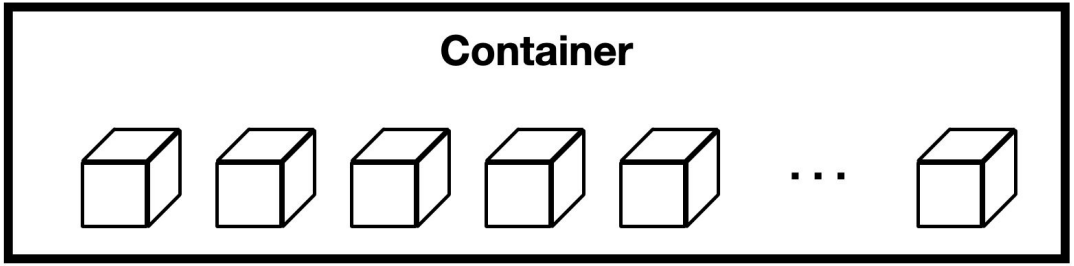
Answer - 3

- Hint: You can directly print the operation and see the result

```
x = False
y = 25
c = True
```

Container

- A container is a class, a data structure, or an ADT
- To easliy store and use the data.
- To imagine, we can think of container as a big box which contains many small boxes



(Fig.) Containers illustration

List (1 / 4) - Assign

- List is a big box which contain many small boxes. The small boxes can be any type.
- To assign list, we use [] to include the value and use , to seperate them.

Code

```
# Assign an empty list to x
x = []
print(x)

# Assign an initial list to y
y = [1, 2, 3]
print(y)

# alternative to assigin an empty
#  list
z = list()
print(z)
```

List (2 / 4) - Get value

- To get the value of small boxes, we should use index. The index **start from 0.**

	1	2	3	4	5	6
Index	**0**	**1**	**2**	**3**	**4**	**5**
Negative index	**-6**	**-5**	**-4**	**-3**	**-2**	**-1**

(Fig.) Index of list

Code

```
# Assign a list to x
x = [1, 2, 3, 4, 5, 6]

# Get value 1
print(x[0])

# Get value 5
print(x[-1])

# Error
print(x[5], x[-6])
```

List (3 / 4) - Properties

- Properties can be regarded as some special built-in function for users to do some operation.
- Below table show some common properties of list. These are **inplace methods** which don't need to be reassign back.

Name	Description
Slicing	x[a:b] can get the value from index a to b - 1
Append	x.append(c) can add c to the end of the list x
Pop out	x.pop() will remove the last value in list x and return the last value.
Remove	x.remove(c) removes the first matching element c from the list.

List (4 / 4) - Properties

Code

```
# Intial the list
x = ["My", "Name", "is", "a"]

# Get the value from index 0 to 2
print(x[0:2+1])

# Remove "a", the last element in list
y = x.pop() # Return "a" back to y
print(y)
print(x)
```

Code

```
# Append "Justin" to the list
x.append("Justin")
print(x)

# Remove "My" in x
x.remove("My")
print(x)
```

Tuple (1 / 2)

- Tuple is also a container which can be store the value.
- To assign a tuple, we use () to include the value and separate them with ",".
- The index rule is same as list.

Code

```
# Define tuple objects
#  and print out the type
x = (1, 2, 3)
print(type(x))
```

Code

```
# To define a tuple with single element
#  add "," after the element
x = (1,) # Tuple
y = (1) # Integer
print(type(x), type(y))
```

Tuple (2 / 2) - Immutable

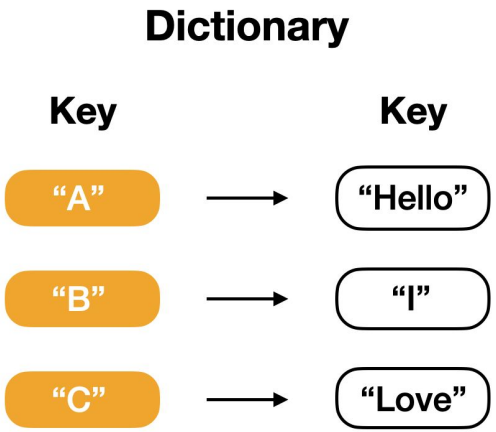
- The content of the tuple couldn't be changed, which means the tuple is immutable.

Code

```
# Define typle objects
x = (1, 2, 3)

# Error
x[0] = 3
x.append(2)
```

Dictionary (1 / 5) - Concept

- Dictionary is a container which include key and value for each elements.
- To obtain a specific value, we use key to search for it.



Dictionary (2 / 5) - Declare

- To declare a dictionary, we use {} to include all the elements. For each element, we use ":" to connect name and value and use "," to separate each element.

Code

```
# Declare dict and print the type
x = {
    "A": "Hello",
    "B": "I",
    "C": "Love"
    }
print(type(x))
```

Dictionary (3 / 5) - Usage

- To get the value, we use x["name"].

Code

```
# Declare list
x = {
    "A": "Hello",
    "B": "I",
    "C": "Love"
    }
# Print out the value
#  using key
print(x["A"], x["B"])
```

Dictionary (4 / 5) - Properties

- Below table show some common properties of dict. The usage of these properties are the same as list.

Name	Description
get	x.get(“name”, c) will return the value of name “name”. If “name” is not exist, return c.
delete	del x[“name”] will remove the key “name” and its value from x.
clear	x.cleare() will clean up x.
set default	x.setdefault(“name”, c) will return the value of name “name”. If “name” is not exist it will return c and add new pair to x.

Dictionary (5 / 5) - Properties

Code

```
# Declare list
x = {
    "A": "Hello",
    "B": "I",
    "C": "Love"
    }

# Get method with exist
#  and not exist name
print(x.get("A", 0))
print(x.get("D", 0))
```

Code

```
# Delete "C"
del x["C"]
print(x)

# Clear x
x.clear()
print(x)

# Set default
print(x.setdefault("A", 1))
print(x)
```

Set (1 /)

- Set items are unordered, immutable and not allow to store duplicate value
- To declare a set, we use {} to include the value and use "," to separate each elements.

Code

```
# Declare a set
x = {1, 1, 2, 2, 3, 4, 5}
print(type(x), x)
```

Length of containers

- To check how many elements in side the containers, we can use built-in function len().

Code

```
# Declare four types of containers
x = [1, 2, 3]
y = (1, 2)
z = {"1": 1}
w = {1, 1, 2, 3}
```

Code

```
print("Length of x:", len(x))
print("Length of y:", len(y))
print("Length of z:", len(z))
print("Length of w:", len(w))
```

Sort the elements

- Often, we want our container sorted in a specific way. We can use the built-in function sorted(). It will return a **list** in increasing order.
- To reverse the order, use **reverse=True**.

Code

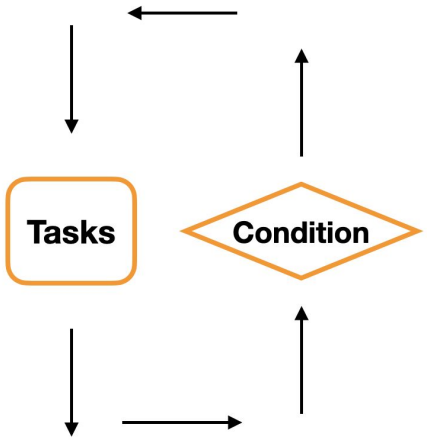
```
# Declare containers
x = [1, 3, 2]
y = (5, 2, 3)
z = {"1": 1, "45": 6, "22": 7}
w = {2, 2, 1, 4, 5}
```

Code

```
print(sorted(x))
print(sorted(y, reverse=True))
print(sorted(z))
print(sorted(w))
```

Loop

- A loop is used for iterating over a sequence or doing the same tasks multiple times until fulfilling some conditions.



For loop (1 / 4) - Concept

- In python, for loop is mainly used to iterate over a sequence like **list, tuple and dictionary**.
- We should add **colon** after the line of for loop and the next line should have **four spaces indentation**.

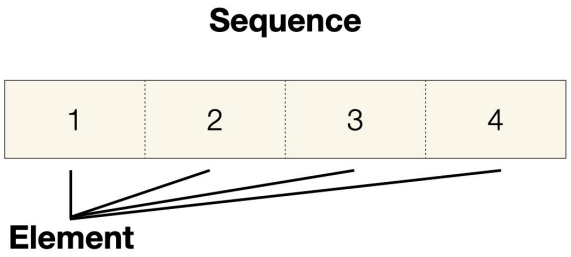
reference

Sequence

```
for element in sequence:
    ...
```

Colon

indentation



For loop (2 / 4) - Example

Code

```
# Create a sequence called words
words = ["Hello", "Python", "I", "Like", "Programming"]

# Using word to reference all the
#  elements in side the sequence
for word in words:
    print(word)
```

For loop (3 / 4) - Range

- Range can create an **integer** sequence
- The rules are shown below

Code

```
range(a) # 0 ~ a-1
range(a, b) # a ~ b-1
range(a, b, c) # a, a+c, ..., b
```

For loop (4 / 4) - Range example

Code

```
# Variable to store the accumulation
res = 0

# Count the accumulation from 1 to 3
for num in range(4):
    res += num

print(res)
```

While loop (1 / 2) - Concept

- While loop often used to do the same tasks multiple times until the condition become False

```
while condition:
    ...
```

Colon

indentation

While loop (2 / 2) - Example

Code

```
# Counter
num = 5

# Count from 5 to 1
while num > 0:
    print(num)
    num -= 1
```

Continue

- Sometimes we want to pass a task during the loop, we can use continue to pass the current task and start the next one.
- It can be used in both while and for loop.

Code

```
# Counter
num = 5

# Count from 5 to 1
#  and pass 2
while num > 0:
    if (num == 2):
        num -= 1
        continue

    print(num)
    num -= 1
```

Break

- In order to make our loop stop early, we can use break to jump out the loop.
- It can be used in both while and for loop.

Code

```
# List for the number
x = [3, 1, 5, 1, 9]

# Print the number
#  until we find 5
for num in x:
    if num == 5:
        break

    print(num)
```

Function – (1 / 7)

- Function is a group of related statements that performs a specific task.
- It help programmers don't repeat themselves.
- Function makes code more readable.

Function - (2 / 7)

- It may be somewhat annoying if we have to type something similar multiple times.

```
res_1 = 3 ** 2 / 3 + 1 * 5
res_2 = 4 ** 2 / 4 + 1 * 5
res_3 = 5 ** 2 / 5 + 1 * 5
...
```

Function - (3 / 7)

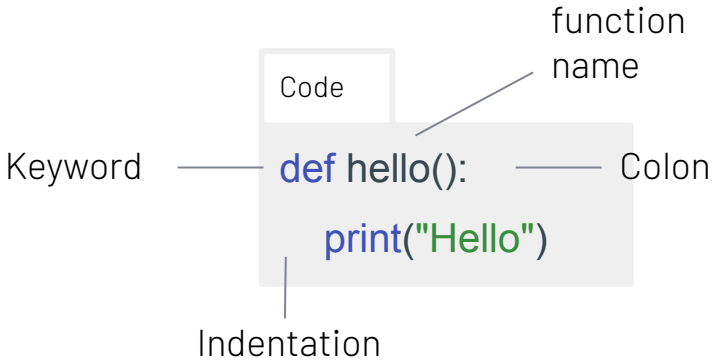
- The better way may be create a function and reused it.

Code

```
# Define the function
def counting(num):
    return num ** 2 / num + 1 * 5

# Using for loop to call the function
#  multiple times
for i in range(3, 6):
    print(counting(i))
```

Function - (4 / 7) - Definition

- To declare a function, we use `def` to define our function.
- The words after `def` is the function name.
- To specify the functionalities, we should leave the indentation for each line in our function body.



Function - (5 / 7) - Call function

- To call the function, we only need to type down it name.

Code

```
# Define a function
def hello():
    print("Hello")

# Call a function
hello()
```

Function - (6 / 7) - Parameters

- We can input the parameters to the function, we only need to specify the parameter name inside ().

Code

```
# Input the parameter
def hello_2(name):
    print("Hello", name)

# Call the function
hello_2("Justin")
```

Function - (7 / 7) - Return value

- We can get the value computed by the function. We only need to use the keyword return.

Code

```
# Get the return value
def compute(number_1, number_2):
    return number_1 ** 2 + number_2 ** 2

# Get and print the
#  return value
x = compute(2, 3)
print(x)
```

Exercise 4

- Design a function that can return the factorial of the input number.

Answer 4

Code

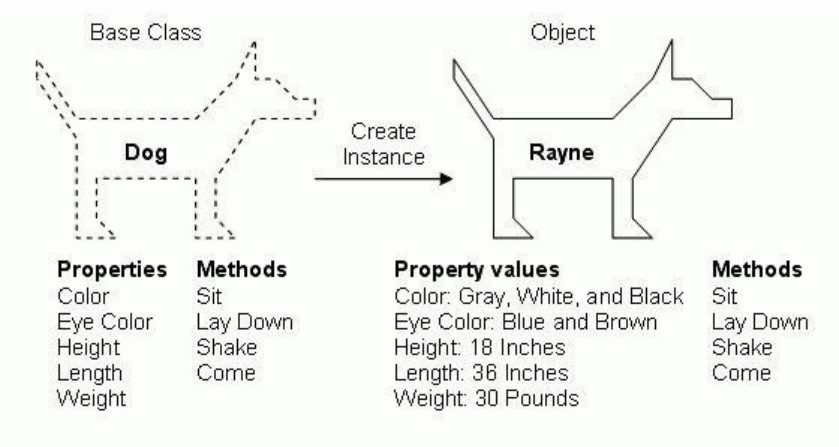
```
def factorial(num):
    # Init the return value
    res = 1

    # Accumualate the value
    for i in range(1, num+1):
        res *= i

    return res
```

Object-oriented (1 / 2)

- Object-oriented programming (OOP) is a method of structuring a program by bundling **related properties and behaviors** into individual objects.

Object-oriented (2 / 2)

- In Python, we can define a **class** that show the common properties and the methods among some **objects**.



(Fig.) Class and object from **Birgitta Edberg**

Class (1 / 7)

- To create a class, use the keyword `class`.
- For example, lets create a class **Number** with properties x = 7.

Code

```
class Number:
    x = 7
```

Class (2 / 7)

- To create an instance, we call it like a function.
- We can access the properties of the class using ".".

Code

```
# Create an Number object
my_number = Number()
print(my_number.x)
```

Code

```
# Reassigned the value of x
my_number.x = 4

# Print the properties of object
print(my_number.x)
# Print the class properties
print(Number.x)
```

Class (3 / 7)

- To initialize the instances, we can define `__init__()` function to do the initialization.

Code

```
# Define a class with __init__ method
class Say:
    def __init__(self):
        print("Hello!")

# Create an instance
my_say = Say()
```

Class (4 / 7)

- We can share the same value of the properties. However, we can assign different values to different instances.
- To specify different values, we use `self` to represent each instance. Also, we can pass the param to the `__init__()` function

Class (5 / 7)

Code

```
# Show the usage of self
class Dog:
    def __init__(self, name, color):
        # Properties of each instance
        self.name = name
        self.color = color

        # Use self. to access the properties
        print("The name of the dog is", self.name)
```

Class (6 / 7)

Code

```
# Create different instances
justin = Dog("Justin", "yello")
jason = Dog("Jason", "white")

# Access the properties
print(justin.color, jason.color)
```

Class (7 / 7)

Code

```
# Class define the properties
#  and method
class Pokemon:
    def __init__(self, name, attack, hp):
        self.name = name
        self.attack = attack
        self.hp = hp

    def popularity(self):
        popularity = self.attack * self.hp
        print("The popularity of", self.name, "is", popularity)
```

Code

```
# Create two instance
zubat = Pokemon("Zubat", 2, 50)
raboot = Pokemon("Raboot", 6, 90)

# Call the class method
zubat.popularity()
raboot.popularity()
```

Part 4
Introduction to Pygame

Pygame

- Pygame is a set of Python modules designed for writing video games.
- Pygame adds functionality on top of the excellent SDL library.

(Fig.) Logo of Pygame

Install Pygame

- Using pip
  - `pip install pygame`
- Using Conda
  - `conda install -c cogsci pygame`
- Test for the installment
  - `python -m pygame.examples.aliens`

Basic element in Pygame (1 / 2)

- Pygame.Surface: Define a rectangle canvas which can shown on the screen.
- Pygame.rect: Define and locate the rectangle area. Can be used to detect the collision.
- Pygame.event: Detect and process the trigger events. Also include the user-defined events.
- Pygame.font: Define the font type and font size.
- Pygame.draw: Used to draw different objects and the background.

Basic element in Pygame (2 / 2)

- Pygame.image: Process I/O of images.
- ...

Learn by coding!

(Fig.) Logo of repl.it