Essential Python 101

- Variables
- data type
- data structure
- function
- control flow
- OOP

```
print("hello world")
```

```
hello world
```

```
print("I am learning Python 101")
```

```
I am learning Python 101
```

```
# commetn
# this is just a note
1+1
2*2
5*3 # python show only last line
```

```
15
```

```
# basic calculation
1+1
2*2
5-3
print(7/2)
print(7//2) # floor devision (ปัดค่าลง)
pow(5,2)
5%2 #Modulo
```

```
3.5
3
1
```

```
# website reference: www.3schools.com/python
```

5 building block

1. variables
2. data types
3. data structures
4. functions
5. control flow
6. OOP (Object Oreinted Program)

```
# assign a variable
my_name = "toy"   # name of variable should be start with lowwer capitle
age = 34
gpa = 3.41  # float type
movie_lover = True #False

# python is all case sensitive
```

```
my_name
```

```
'toy'
```

```
print(age, gpa, movie_lover, my_name)
```

```
34 3.41 True toy
```

```
# over write a value
age = 22
new_age = age - 12
print(age, new_age)
```

```
22 10
```

```
s23_price = 30000
discount = 0.15
new_s23_price = s23_price *(1-discount)

print(new_s23_price)
```

```
25500.0
```

```
# remove variable
del new_s23_price
```

```
# count variable
age = 34
age += 1 # age = age +1
print(age)
```

```
35
```

```
# count variable
age = 34
age += 1
age += 1
age -= 1
age *= 2
age /= 2
print(age)
```

```
35.0
```

```
# data types
# int float str bool
age = 38
pda = 2.75
school = "Chulalongkorn"
movier_lover = True
```

```
# check data type with type()
print(type(age))
print(type(gpa))
print(type(school))
print(type(bool))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'type'>
```

```
# convert type
# str() int() flo() bool()
x = 100
x = str(x)
print(x, type(x))
```

```
100 <class 'str'>
```

```
y = True #T=1, F=0
y = int(y)
print(y, type(y))
```

```
1 <class 'int'>
```

```
z = 1
z = bool(z)
print(z,type(z))
```

```
True <class 'bool'>
```

```
text = "I'm learning Python"
text2 = '"Haha"'
print(text, text2)
```

```
I'm learning Python "Haha"
```

```
text = "hello"
print(text +text +text,text*4)
```

```
hellohellohello hellohellohellohello
```

```
# type hint
age: int = 34  #: assingn data type
print(age,type(age))
```

```
34 <class 'int'>
```

```
# type hint
age:int = 38
my_name: str = "Tawat"
pda: float = 2.75
seafood: bool = True
```

3. Funtion

```
# function
print("hello", "world")
pow(5,2)    # built in function

# create function: greeting()
def greeting(name = "John", location = "London"):  #set variable and assign defaul
    print("hello " + name)
```

```
    print("He is at " + location)

```

```
hello world
```

```
# test function greeting
greeting()
```

```
hello John
He is at London
```

```
greeting("Tawat", "Bangkok")
```

```
hello Tawat
He is at Bangkok
```

```
greeting(location = "Bangkok", name = "Tawat") # swap location
```

```
hello Tawat
He is at Bangkok
```

```
def add_two_nums(num1, num2):
    print("hello world!")
    print("Hey")
    return num1 + num2
```

```
result = add_two_nums(5,15)
print(result)
```

```
hello world!
Hey
20
```

```
def add_two_nums(a:int, b:int) -> int:
    return a+b
```

```
add_two_nums(5,6)
```

```
11
```

```
# work with string
text = "hello world"
long_text = """
this is a
very long text
this is a new line"""
```

```
print(text)
print(long_text)
```

```
hello world

this is a
very long text
this is a new line
```

```
# string template : fstrings
my_name = "John Wick"
location = "London"

text = f"Hi! myname is {my_name} and I live in {location}"
print(text)
```

```
Hi! myname is John Wick and I live in London
```

```
"Hi! myname is {} ,and I live in {}".format(my_name, location)
```

```
'Hi! myname is John Wick ,and I live in London'
```

```
text = "a duck walks into a bar"
print(text)
```

```
a duck walks into a bar
```

```
len(text)
```

```
23
```

```
# slicing, index start with 0
print(text[0], text[22], text[-1])
```

```
a r r
```

```
text
```

```
'a duck walks into a bar'
```

```
# up to, but not include
text[7:12]
```

```
'walks'
```

```
text[13:17]
```

```
'into'
```

```
text[7:]
```

```
'walks into a bar'
```

```
text[-3:]
```

```
'bar'
```

```
# string is immutable
name = "Python" # -> Cython
name = "C" +name[1:]
print(name)
```

```
Cython
```

```
text = "a duck walks into a bar"
# function vs. method
# string methods -> Function ที่ใช้กับ data ตระกูล String
text.upper()
```

```
'A DUCK WALKS INTO A BAR'
```

```
text
```

```
'a duck walks into a bar'
```

```
text = text.upper()
print(text)
```

```
A DUCK WALKS INTO A BAR
```

```
text = text.lower()
print(text)
```

```
a duck walks into a bar
```

```
text.replace("duck", "lion")
```

```
'a lion walks into a bar'
```

```
words = text.split(" ")
print(words, type(words))
```

```
['a', 'duck', 'walks', 'into', 'a', 'bar'] <class 'list'>
```

```
" ".join(words)
```

```
'a duck walks into a bar'
```

```
"-".join(words)
```

```
'a-duck-walks-into-a-bar'
```

Data Strunture

1.list[]

2.tuple()

3.dictionary{}

4.set{unique}

```
# Data Strunture
## 1.list[]
## 2.tuple()
## 3.dictionary{}
## 4.set{unique}

# list
shopping_items = ["banana", "egg", "milk"]
print(shopping_items[0])
print(shopping_items[1])
print(shopping_items[1:])
print(len(shopping_items))
```

```
banana
egg
['egg', 'milk']
3
```

```
# list is mutable
shopping_items = ["banana", "egg", "milk"]

shopping_items[0] = "Pineapple"
shopping_items[1] = "Cheese"

print(shopping_items)
```

```
['Pineapple', 'Cheese', 'milk']
```

```
# list methods
shopping_items.append("egg")
print(shopping_items)
```

```
['Pineapple', 'Cheese', 'milk', 'egg']
```

```
# sort items
shopping_items.sort(reverse = True)
print(shopping_items)
```

```
['milk', 'Pineapple', 'Cheese']
```

```
scores = [90, 88, 85, 92, 75]
print(len(scores), sum(scores), min(scores), max(scores))
```

```
5 430 75 92
```

```
def mean(scores):
    return sum(scores)/ len(scores)
```

```
mean(scores)
```

```
86.0
```

```
sum(scores)/ len(scores)
```

```
86.0
```

```
# remove last item in list
shopping_items.pop()
shopping_items
```

```
['Pineapple', 'Cheese', 'milk']
```

```
# add item into list
shopping_items.append("egg")
shopping_items
```

```
['Pineapple', 'Cheese', 'milk', 'egg']
```

```
shopping_items.remove("milk")
shopping_items
```

```
['Pineapple', 'Cheese', 'egg']
```

```
# .insert()
shopping_items.insert(1,"milk")
```

```
shopping_items
```

```
['Pineapple', 'milk', 'Cheese', 'egg']
```

```
#list + list
items1 = ['egg', 'milk']
items2 = ['banana', 'bread']

print(items1 + items2)
```

```
['egg', 'milk', 'banana', 'bread']
```

```
# tuple () is immutable
tup_items = ('egg', 'bread', 'pepsi')
tup_items
```

```
('egg', 'bread', 'pepsi')
```

```
tup_items.count('egg')
```

```
1
```

```
## user & password
## student1, student2
s1 = ("id001", "123456")
s2 = ("idoo2", "654321")
user_pw = (s1,s2)

print(user_pw)
```

```
(('id001', '123456'), ('idoo2', '654321'))
```

```
# tuple unpacking
username, password = s1

print(username, password)
```

```
id001 123456
```

```
# tuple unpacking 3 values
name, age, _ = ("John Wick", 42, 3.98)
```

```
3
4 print(name, age)
```

```
John Wick 42
```

```
1 # set {unique}
2 courses = ["Python", "Python", "R", "SQL", "SQL", "SQL"]
```

```
1 courses
```

```
['Python', 'Python', 'R', 'SQL', 'SQL', 'SQL']
```

```
1 set(courses)
```

```
{'Python', 'R', 'SQL'}
```

```
1 # dictionary key: value pairs
2 course = {
3     "name": "Data Science Bootcamp",
4     "duration": "4 months",
5     "students": 200,
6     "replay": True,
7     "skills": ["google Sheets", "SQL", "R", "Python",
8                "stats", "ML", "Dashboard", "Data Transformation"]
9 }
```

```
1 course
```

```
{'name': 'Data Science Bootcamp',
 'duration': '4 months',
 'students': 200,
 'replay': True,
 'skills': ['google Sheets',
  'SQL',
  'R',
  'Python',
  'stats',
  'ML',
  'Dashboard',
  'Data Transformation']}
```

```
1 course["replay"]
```

```
True
```

```
course["start_time"] = "9am"

course["language"] = "Thai"
```

```
course
```

```
{'name': 'Data Science Bootcamp',
 'duration': '4 months',
 'students': 200,
 'replay': True,
 'skills': ['google Sheets',
  'SQL',
  'R',
  'Python',
  'stats',
  'ML',
  'Dashboard',
  'Data Transformation'],
 'start_up': '9am',
 'language': 'Thai'}
```

```
# delete
# del course["start_up"]
course["replay"] = False
```

```
course
```

```
{'name': 'Data Science Bootcamp',
 'duration': '4 months',
 'students': 200,
 'replay': False,
 'skills': ['google Sheets',
  'SQL',
  'R',
  'Python',
  'stats',
  'ML',
  'Dashboard',
  'Data Transformation'],
 'start_up': '9am',
 'language': 'Thai'}
```

```
course["skills"][-3:]
```

```
['ML', 'Dashboard', 'Data Transformation']
```

```
course.keys()
```

```
dict_keys(['name', 'duration', 'students', 'replay', 'skills', 'start_up',
'language'])
```

```
1 list(course.keys())
```

```
['name', 'duration', 'students', 'replay', 'skills', 'start_up', 'language']
```

```
1 list(course.values())
```

```
['Data Science Bootcamp',
 '4 months',
 200,
 False,
 ['google Sheets',
  'SQL',
  'R',
  'Python',
  'stats',
  'ML',
  'Dashboard',
  'Data Transformation'],
 '9am',
 'Thai']
```

```
1 list(course.items())
```

```
[('name', 'Data Science Bootcamp'),
 ('duration', '4 months'),
 ('students', 200),
 ('replay', False),
 ('skills',
  ['google Sheets',
   'SQL',
   'R',
   'Python',
   'stats',
   'ML',
   'Dashboard',
   'Data Transformation']),
 ('start_up', '9am'),
 ('language', 'Thai')]
```

```
1 course["replay"]
```

```
True
```

```
1 # Recap
2 # list, dictionary = mutable
3 # tuple, string = immutable
4
```

Control Flow

```
# final exam 150 questions, pass >= 120
def grade(score):
    if score >=120:
        return "Excellent"
    elif score >= 100:
        return "Good"
    elif score >= 80:
        return "OK"
    else:
        return "Need to read more!"
```

```
result = grade(95)
print(result)
```

```
OK
```

```
# use  adn, or, not in condition
# course == data science, score >=80 passed
# course == english, score >=70 passed
def grade(course, score):
    if course == "english" and score >= 70:
        return "passed"
    elif course == "data science" and score >= 80:
        return "passed"
    else:
        return "failed"
```

```
grade("data science", 81)
```

```
'passed'
```

```
not True
```

```
False
```

```
# for Loop
# if score >= 80, passes
def grading_all(scores):
    new_scores = []
    for score in scores:
        new_scores.append(score+2)
    return new_scores
```

```
grading_all([75, 88, 90, 95, 52])
```

```
[77, 90, 92, 97, 54]
```

```
# list comprehension
scores = [75, 88, 90, 95, 52]
[s*2 for s in scores]
```

```
[150, 176, 180, 190, 104]
```

```
friends = ["toy", "ink", "bee", "zue", "yos"]
[ f.upper() for f in friends]
```

```
['TOY', 'INK', 'BEE', 'ZUE', 'YOS']
```

```
# while loop
count = 0

while count < 5:
    print("hello")
    count += 1
```

```
hello
hello
hello
hello
hello
```

```
# chatbot for fruit order
user_name = input("What is your name?")
```

```
What is your name?John Wick
```

```
user_name
```

```
'John Wick'
```

```
def chatbot():
    fruits = []
    while True:
        fruit = input("What fruit do you want to order?")
        fruits.append(fruit)
        if fruit == "exist":
            return fruits
```

```
chatbot()
```

```
What fruit do you want to order?banana
What fruit do you want to order?orange
What fruit do you want to order?grape
What fruit do you want to order?rawberry
What fruit do you want to order?exist
['banana', 'orange', 'grape', 'rawberry', 'exist']
```

```
# HW01 - chatbot to order pizza
# HW02 - pao ying chup
```

```
# OOP = Object Oriented Programming
# DOg Class
```

```
class Dog:
    def __init__(self, name, age, breed): #dunder -> Double Underscore
        self.name = name
        self.age = age
        self.breed = breed
```

```
dog1 = Dog("ovaltine",2, "chihuahua")
dog2 = Dog("milo", 3, "german chepherd")
dog3 = Dog("pepsi", 3.3,"Thai" )
```

```
dog1
```

```
<__main__.Dog at 0x7f77a10c99d0>
```

```
print(dog1.name, dog1.age, dog1.breed )
```

```
ovaltine 2 chihuahua
```

```
class Employee:
    def __init__(self, id, name, dept, pos):
        self.id = id
        self.name = name
        self.dept = dept
        self.pos = pos # position
    def hello(self):
        print(f"Hello! my name is {self.name}")
    def work_hours(self,hours):
        print(f"{self.name} work for {hours} hours.")
    def change_dep(self, new_dept):
        self.dept = new_dept
        print(f"{self.name}is now in {self.dept}.")

```

```
1 emp1 = Employee(1, "John", "Finance", "Financial Analyst")
```

```
1 print(emp1.name, emp1.pos)
```

```
    John Financial Analyst
```

```
1 emp1.hello()
```

```
    Hello! my name is John
```

```
1 emp1.work_hours(5.5)
```

```
    John work for 5.5 hours.
```

```
1 emp1.change_dep("data science")
```

```
    Johnis now in data science.
```

```
1 emp1.dept
```

```
    'data science'
```

```
1 # H03 - create new ATm class
```