Essential Python 101

- variables
- data types
- data structure
- function
- control flow
- oop

```
# comment
1+1
print(7//2) #floor division
print(7/2)
pow(5, 3)
abs(-666)
#modulo
5%2
```

```
3
3.5
1
```

```
#assign a varible
my_name = "toy"
age = 34
movie_lover = True #False
```

```
print(age, movie_lover)
```

```
34 True
```

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

print(new_price)
```

```
25500.0
```

```
#remove variable
del s23_price
```

```
#count variable
age = 34
age += 1
age += 1
age += 1
age -= 2
age *= 2
age /= 3
print(age)
```

```
23.333333333333332
```

```
#data types
#int float str bool
```

```
age = 34
gpa = 3.41
school = "kmitl"
movie_lover = True
```

```
print( type(age))
print( type(gpa))
print( type(school))
print( type(movie_lover))
```

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

```python
#convert type
x = 100
x = str(x) #convert to string
print(x, type(x))
```

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

```python
y = True #T=1, F=0
y = int(y)
print(y, type(y)) #convert boolean to int
```

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

```python
z = 1
z = bool(z)
print(z, 'bool')
```

```
True bool
```

```python
age = 34
print(age, age*2, age/2)
```

```
34 68 17.0
```

```python
text = "I'm learning Python"
text2 = "hello"
print (text+text2, text*4)
```

```
I'm learning Pythonhello I'm learning PythonI'm learning PythonI'm learning PythonI'm learning Python
```

```python
# type int
age: int = 34
my_name: str ="Toy"
gpa: float = 3.41
seafood: bool = True
print(age, type(age))
```

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

```python
#function
# greeting

def greeting(name="Tip", location="Oslo"):
    print("Hello " + name)
    print("She is at " + location)
```

```python
greeting(location="Japan", name="Win")
greeting(location="Washington", name = "Toy")
```

```
Hello Win
She is at Japan
Hello Toy
She is at Washington
```

```python
def add_two_num(num1, num2):
    print("hello world")
    return(num1+num2) #อะไรหลัง return จะจบการทำงาน
```

```python
result = add_two_num(2,4)
print(result)
```

```
hello world
6
```

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

text = f"Hi! my name is {my_name} and I live in {location}"

print(text)
```

```
Hi! my name is John Wick and I live in London
```

```
1 # function designed for string (string methods)
2 text = "a duck walks into a bar"
3 print(text)
```

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

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

```
a r r
```

```
1 text[2:6]
```

```
'duck'
```

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

```
'walks'
```

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

```
'bar'
```

```
1 text[7: ]
```

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

```
1 #string is imutable #ถ้าประกาศตัวแปรเป็น string จะเปลี่ยนเลยไม่ได้
2 name ="Python" # -> Cython
3 name = "C" + name[1:]
4 print(name)
```

```
Cython
```

```
1 name = "Python"
2 ̃name = "Cython"
3 print(name)
```

```
  File "<ipython-input-6-3398b110ee4c>", line 2
    ̃name = "Cython"
     ^
SyntaxError: invalid character '̃' (U+0E37)
```

SEARCH STACK OVERFLOW

```
1 text = "a duck walks into a bar"
```

```
1 #function vs. methods: method is function for specific object
2 #string method
3 #string is imutable
4 text.upper()
5
```

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

```
1 text.lower()
```

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

```
1 text.title()
```

```
'A Duck Walks Into A Bar'
```

```
1 text.lower()
```

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

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

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

```
1 words = text.split(" ") #split white space
2 print(words, type(words))
```

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

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

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

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

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

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

```
#list is mutable (can update value)
shopping_list = ['egg', 'milk', 'bread']
print(shopping_list)
```

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

```
print(shopping_list[1])
shopping_list[0] ="Pineapple"
print(shopping_list)
```

```
milk
['Pineapple', 'milk', 'bread']
```

```
print(len(shopping_list[1]))
```

```
4
```

```
shopping_list = ['egg', 'milk', 'bread']
```

```
shopping_list.append('orange juice') #add value on the right
print(shopping_list)
```

```
['egg', 'milk', 'bread', 'orange juice']
```

```
#sort items: ascending order
shopping_list.sort()
print(shopping_list)
```

```
['bread', 'egg', 'milk', 'orange juice']
```

```
#sort descending order
shopping_list.sort(reverse=True)
print(shopping_list)
```

```
['orange juice', 'milk', 'egg', 'bread']
```

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

```
score = [90,88,34,23]
print(sum(score), min(score), max(score), len(score), mean(score))
```

```
235 23 90 4 58.75
```

```
sum(score)/len(score) #average
```

```
58.75
```

```
# remove last item in the list
# list method .pop()
shopping_list.pop()
print(shopping_list)
```

```
['orange juice', 'milk']
```

```
# remove specific item
shopping_list.remove("milk")
```

```
3 shopping_list
```

```
['orange juice']
```

```
1 # .insert()
2 shopping_list.insert(1, "Coke")
3 shopping_list
```

```
['orange juice', 'Coke']
```

```
1 # list+list
2 items1 = ['egg', 'milk']
3 items2 = ['banana', 'bread']
4
5 print(items1+items2)
```

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

```
1 #tuple items is imutable (keep value only-cannot update)
2 tup_item = ('egg', 'bread', 'egg', 'egg')
3 tup_item
```

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

```
1 tup_item.count('egg')
```

```
3
```

```
1 #usernanme password (tuple)
2 s1 = ("id001", "1234")
3 s2 = ("id002", "6543")
4 user_pw = (s1,s2)
5
6 print(user_pw)
```

```
(('id001', '1234'), ('id002', '6543'))
```

```
1 #tuple unpacking
2 username, password = s1
3
4 print(username, password)
```

```
id001 1234
```

```
1 #tuple unpacking 3 values
2 name, age, gpa = ("John Wick", 42, 3.98)
3 print(name, age)
```

```
John Wick 42
```

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

```
John Wick 42
```

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

```
1 set(course)
```

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

```
1 #dictionary key: value pairs (mutable)
2 course ={
3     "name": "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': 'Bootcamp',
 'duration': '4 months',
 'students': 200,
 'replay': True,
 'skills': ['Google Sheets',
  'SQL',
  'R',
  'Python',
  'Stats',
  'ML',
  'Dashboard',
  'Data transformation']}
```

```
course["name"]
```

```
'Bootcamp'
```

```
course["skills"]
```

```
['Google Sheets',
 'SQL',
 'R',
 'Python',
 'Stats',
 'ML',
 'Dashboard',
 'Data transformation']
```

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

```
True
```

```
#add key
course["start_time"] = "9am"

course["language"] = "Thai"

course
```

```
{'name': 'Bootcamp',
 'duration': '4 months',
 'students': 200,
 'replay': True,
 'skills': ['Google Sheets',
  'SQL',
  'R',
  'Python',
  'Stats',
  'ML',
  'Dashboard',
  'Data transformation'],
 'start_time': '9am',
 'language': 'Thai'}
```

```
#delete key
del course["language"]
course
```

```
{'name': 'Bootcamp',
 'duration': '4 months',
 'students': 200,
 'replay': True,
 'skills': ['Google Sheets',
  'SQL',
  'R',
  'Python',
  'Stats',
  'ML',
  'Dashboard',
  'Data transformation'],
 'start_time': '9am'}
```

```
#update key
course["replay"] = False
course
```

```
{'name': 'Bootcamp',
 'duration': '4 months',
 'students': 200,
 'replay': False,
 'skills': ['Google Sheets',
  'SQL',
  'R',
```

```
      'Python',
      'Stats',
      'ML',
      'Dashboard',
      'Data transformation'],
     'start_time': '9am'}
```

```
1 course["skills"][0:3]
```

```
['Google Sheets', 'SQL', 'R']
```

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

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

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

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

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

```
['name', 'duration', 'students', 'replay', 'skills', 'start_time']
```

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

```
['Bootcamp',
 '4 months',
 200,
 False,
 ['Google Sheets',
  'SQL',
  'R',
  'Python',
  'Stats',
  'ML',
  'Dashboard',
  'Data transformation'],
 '9am']
```

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

```
[('name', 'Bootcamp'),
 ('duration', '4 months'),
 ('students', 200),
 ('replay', False),
 ('skills',
  ['Google Sheets',
   'SQL',
   'R',
   'Python',
   'Stats',
   'ML',
   'Dashboard',
   'Data transformation']),
 ('start_time', '9am')]
```

```
1 course.get("replay")
```

```
False
```

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

```
False
```

```
1 #recap
2 #list, dictionary = mutable
3 #tuple, string = imutable
```

```
1 #control flow
2 #if for while
```

```
1 #final exam 150 questions, pass >=120
2 score = 125
3 if score >= 120:
4     print("passed")
5 else:
6     print("failed")
```

```
passed
```

```
def grade (score):
    if score >= 120:
        return("passed")
    else:
        return("failed")
```

```
result = grade(144)
print(result)
```

```
passed
```

```
def grade (score):
    if score >= 120:
        return("Excellent")
    elif score >= 100:
        return "Good"
    elif score >= 80:
        return "Okay"
    else:
        return "Need to read more!"
```

```
result = grade(85)
print (result)
```

```
Okay
```

```
# use and, or in condition
# course == data science, score >= 88 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
scores = [88, 90, 75]

for score in scores:
    print(score-2)

```

```
86
88
73
```

```
new_scores = []

for score in scores:
    new_scores.append(score-2)
print(new_scores)
```

```
[86, 88, 73]
```

```
#grading all
def grading_all(scores):
    new_scores = []
    for score in scores:
        new_scores.append(score-2)
    return new_scores
```

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

```
[73, 86, 88, 50]
```

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

```
[176, 180, 150]
```

```
#list comprehension
#for loop
friends = ["toy", "ink", "bee", "pui"]

[f.upper() for f in friends]
```

```
['TOY', 'INK', 'BEE', 'PUI']
```

```
#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? Johnwick
```

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

        if fruit=='exit':
            return fruits
        fruits.append(fruit)
```

```
chatbot()
```

```
What fruit do you want to order? apple
What fruit do you want to order? strawberry
What fruit do you want to order? exit
['apple', 'strawberry']
```

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

```
age = int( input("How old are you? "))
```

```
How old are you? 34
```

```
type(age)
```

```
int
```

```
#OOP -Object Oriented Programming
#Dog class
```

```
class Dog:
    def __init__(self, name, age, breed): #__ dunder
        self.name = name #int initiate
        self.age = age
        self.breed = breed
```

```
dog1 = Dog("ovaltine", 4, "chihuahua")
dog2 = Dog("milo", 3, "poodle")
dog3 = Dog("pepsi", 3.5, "bulldog")
```

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

```
ovaltine 4 chihuahua
```

```
class Employee:
    def __init__(self, id, name, dept, pos): #__ dunder
        self.id = id #int initiate
        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} works for {hours} hours.")

    def change_dept(self, new_dept):
        self.dept = new_dept
        print(f"{self.name} is now in {new_dept}")
```

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

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

```
John Analyst
```

```
emp1.hello()
```

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

```
emp1.work_hours(10)
```

```
John works for 10 hours.
```

```
emp1.change_dept("Data Science")
```

```
John is now in Data Science
```

```
#object: attribute => name, id, pos
#object: method => hello(), change_dept()
```

```
#HW03 - create new ATM class (5 methods)

class ATM:
    def __init__(self, name, bank, balance):
        self.name = name
        self.bank = bank
        self.balance = balance
    def deposit(self, amt):
        self.balance += amt

scb = ATM("toyeiei", "scb", 500)
print(scb.balance)

scb.deposit(100)
print(scb.balance)
```

```
500
600
```

```
1
```

✓ 0s completed at 10:50 PM