Python End to End

- f-strings
- string methods
- nested dictionary
- functions, try-catch
- lambda
- list comprehension
- OOP & class (Object Oriented Programming)
- ATM mini project

```
# f-strings template

my_name = "Toy"
my_age = 33
my_fav_language = "R"

# long string """
text = f"""Hello my name is {my_name}. I'm {my_age} years old
and my favorite language is {my_fav_language}."""

print(text)
```

```
Hello my name is Toy. I'm 33 years old
and my favorite language is R.
```

```
# define a new function
def greeting(name):
    return f"Hello! {name}"

# test function
greeting("David")
```

```
# string methods
# what is method?
# String is immutable
my_name = "Python"

my_new_name = "S" + my_name[1:]

print(my_new_name)
```

Sython

```
# string method
text = "a duck walks into a bar"
```

```
text = text.replace('duck', 'tiger')
```

```
print(text)
```

```
a tiger walks into a bar
```

```
list_text = text.split(" ")
print(list_text)
```

```
['a', 'tiger', 'walks', 'into', 'a', 'bar']
```

```
# join words in a list using " "
" ".join(list_text)
```

```
# nested dictionary
# key-value pairs
player_01 = {
    "fname": "David",
    "lname": "Beckham",
    "clubs": ["Man United", "Real Madrid"]
}

player_02 = {
    "fname": "Cristiano",
    "lname": "Ronaldo",
    "clubs": ["Man United", "Real Madrid", "Juventus"],
    "prize": True,
    "gamer": True
}
```

```
all_players = {
    "01": player_01,
    "02": player_02
}

all_players
```

```
{'01': {'clubs': ['Man United', 'Real Madrid'],
  'fname': 'David',
  'lname': 'Beckham'},
 '02': {'clubs': ['Man United', 'Real Madrid', 'Juventus'],
```

```
      'fname': 'Cristiano',
      'gamer': True,
      'lname': 'Ronaldo',
      'prize': True}}
```

```
# JSON == Python dict
# JavaScript Object Notation

all_players["01"]["clubs"][1]
```

```
player_01.get("firstName", "Not Found")
```

```
# loop through dictionary
for key, value in player_01.items():
    print( f"{key}: {value}" )
```

```
fname: David
lname: Beckham
clubs: ['Man United', 'Real Madrid']
```

```
# list of values
balls = ["red", "green", "blue", "blue", "green"]

def count_ball(balls):
    result = {}
    kk = "Toy" # local variable
    for ball in balls:
        if ball in result:
            result[ball] += 1
        else:
            result[ball] = 1
    return result

result = count_ball(balls)
print(result)
```

```
{'red': 1, 'green': 2, 'blue': 2}
```

```
result
```

```
{'blue': 2, 'green': 2, 'red': 1}
```

```
# try catch syntax
# manage error message in Python
```

```
x = 10

try:
    result = x / 0
    print(result)
except NameError:
    print("X is not defined")
except ZeroDivisionError:
    print("Cannot divided by zero")
finally:
    print("The End!")
```

```
Cannot divided by zero
The End!
```

```
# lambda function / Refactoring
greeting = lambda name: print(f"Hello! {name}")
```

```
greeting("Toy")
```

```
Hello! Toy
```

```
def greeting(name):
    print(f"Hello! {name}")
```

```
# common pattern when we use lambda function
values = [1, 3, 5, 10, 12]

# 1. list comprehension
new_values = [ value*2 for value in values ]

print(new_values)
```

```
[2, 6, 10, 20, 24]
```

```
# 2. lambda function + map()
# anonymous function
result = list(map(lambda number: number/2, values))
print(result)
```

```
[0.5, 1.5, 2.5, 5.0, 6.0]
```

```
# lambda function
divided_by_two = lambda number: number/2
```

```
divided_by_two(10)
```

```
5.0
```

```
1
```

OOP

Object Oriented Programming

```
# create a new class
class Dog:
    # attributes
    def __init__(self, name, age, breed):
        self.name = name
        self.age = age
        self.breed = breed

    # sitting
    def sitting(self):
        print("I'm sitting now!")

    # introduce yourself
    def hello(self):
        print( f"Hello! my name is {self.name}. My breed is {self.breed}" )
```

```
my_dog = Dog(name = "Kuma", age = 2, breed = "Pomeranian")
```

```
my_dog.hello()
```

```
Hello! my name is Kuma. My breed is Pomeranian
```

```
1
```

ATM mini project

```
import random

class ATM:
    def __init__(self, account_name, bank_name):
        self.account_name = account_name
        self.bank_name = bank_name
        self.balance = 0

    # 1. deposit
    def deposit(self, amount):
```

```
        self.balance += amount
        print(f"Your balance is {self.balance}.")

    # 2. withdraw
    def withdraw(self, amount):
        if amount > self.balance:
            print("Not enough money.")
        else:
            self.balance -= amount
            print(f"Your balance is {self.balance}.")

    # 3. check balance
    def check_balance(self):
        print(f"Your balance: {self.balance}")

    # 4. mobile banking
    def mobile_withdraw(self, amount):
        OTP = random.randint(10000, 99999)
        print(OTP)

        # get OTP input from user
        user_input = int(input("Your OTP: "))
        if user_input == OTP:
            if amount > self.balance:
                print("Not enough money.")
            else:
                self.balance -= amount
                print(f"Your balance is {self.balance}")
        else:
            print("Your OTP is incorrect. Please try again!")
```

```
1 my_atm = ATM("Toy", "SCB")
```

```
1 my_atm.deposit(500)
```

```
Your balance is 800.
```

```
1 my_atm.withdraw(1000)
```

```
Not enough money.
```

```
1 my_atm.check_balance()
```

```
Your balance: 800
```

```
1 my_atm.mobile_withdraw(450) # left 350 THB
```

```
41293
```

```
    Your OTP: 41293
    Your balance is 350


1 # Homework
2 # Create new class : Restaurant
3 # Minimum 5 methods


1
```