NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Seminar 4: Function

Function

- **Function** is a set of codes which can be repeatable when the function name is called.
- We can **use functions** which were written by others.
- Or, we can **make** the **functions ourselves** and use it.

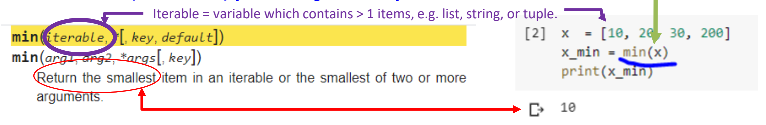
2

1. Using functions

- Know the **name** of function and call it within your program.
- Python has a standard library that comes with Built-in functions. That means, you can use those functions readily.
  - See: https://docs.python.org/3/library/functions.html
- Basic functions: print(), input()
- Mathematical functions: abs(), max(), min(), sum(), round(), pow(), len()
- Type conversion functions: int(), float(), str(), list()

1. Using functions

- Mathematical functions: abs(), max(), min(), sum(), round(), pow(), len()
- Example 1:
  - We first create a variable x with multiple number -> **list**. (List starts with square bracket followed by a series of comma separated numbers and close with closing square bracket)
  - With the list of number, we pass that information into a function called **min(x)** to get the minimum number among that list of number. This passing of information to function is called **input argument.**
  - The **min()** function will then **return** us a final number which is the smallest among the list
  - See https://docs.python.org/3/library/functions.html#min


Iterable = variable which contains > 1 items, e.g. list, string, or tuple.

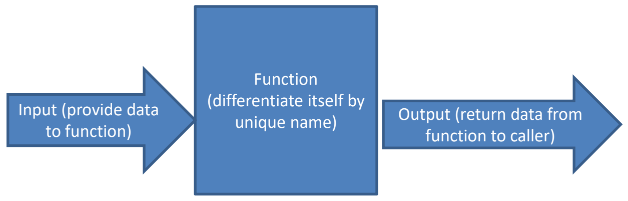
```
min(iterable, *[, key, default])
min(arg1, arg2, *args[, key])
```

Return the smallest item in an iterable or the smallest of two or more arguments.

```
[2] x  = [10, 20, 30, 200]
    x_min = min(x)
    print(x_min)

10
```

2. Making Function


Input (provide data to function)
Function (differentiate itself by unique name)
Output (return data from function to caller)

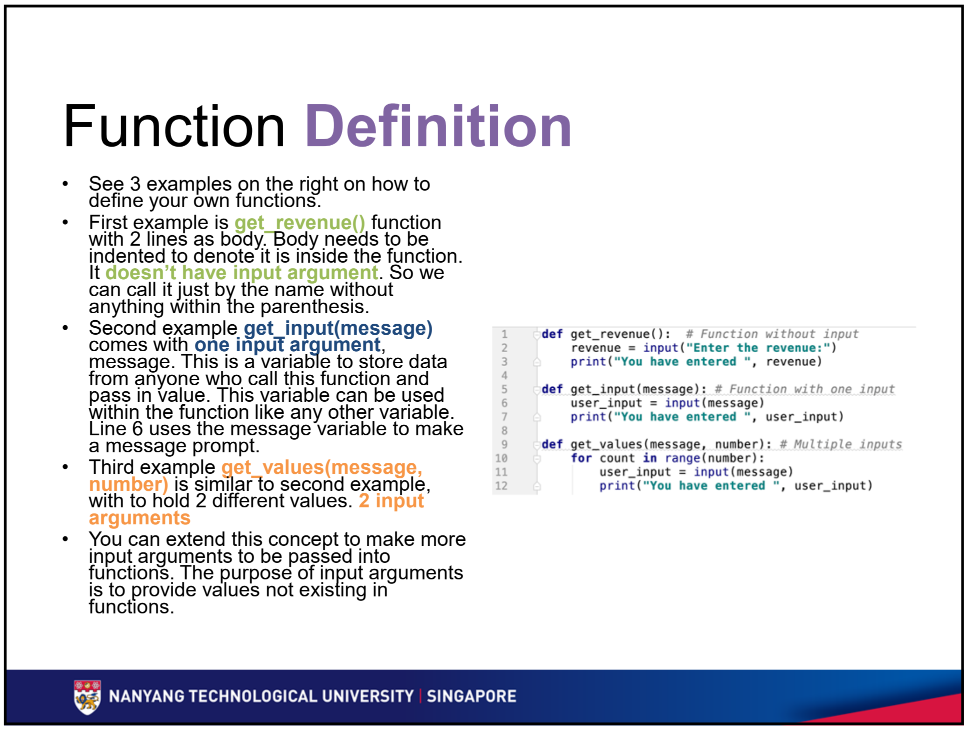
2. Making Function -> Define

- Function is a group of statements.
- To perform a single task.
- Two parts to a function: **define** and usage
  - **Define**:

```
def function_name():
    body
```

  - Usage:

```
function_name()
```

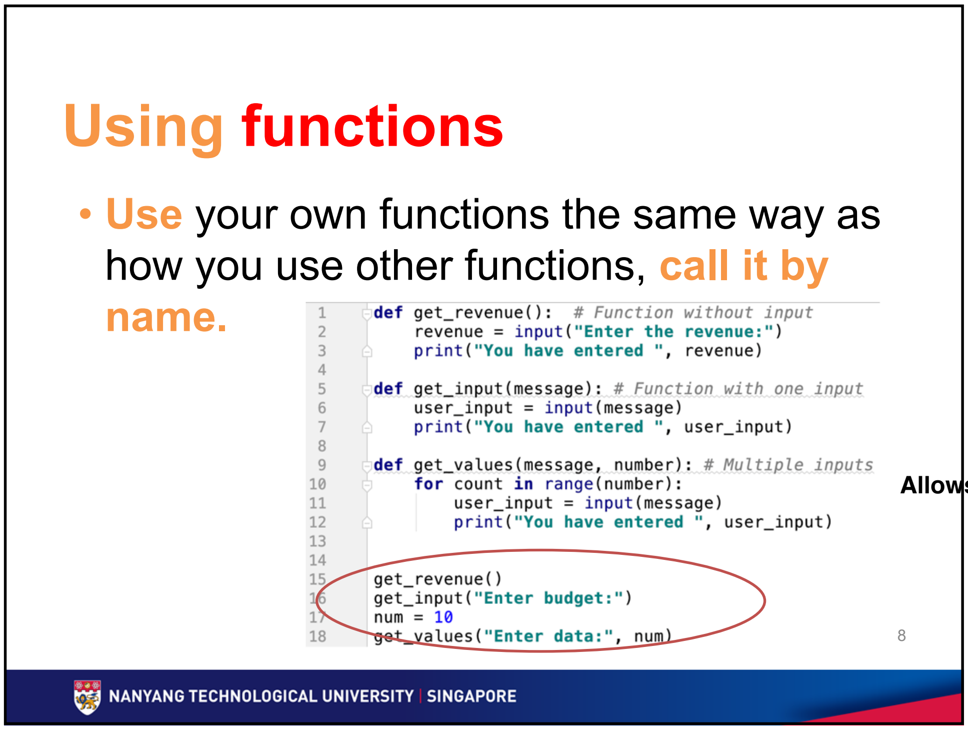


Function Definition

- See 3 examples on the right on how to define your own functions.
- First example is get_revenue() function with 2 lines as body. Body needs to be indented to denote it is inside the function. It doesn't have input argument. So we can call it just by the name without anything within the parenthesis.
- Second example get_input(message) comes with one input argument, message. This is a variable to store data from anyone who call this function and pass in value. This variable can be used within the function like any other variable. Line 6 uses the message variable to make a message prompt.
- Third example get_values(message, number) is similar to second example, with to hold 2 different values. 2 input arguments
- You can extend this concept to make more input arguments to be passed into functions. The purpose of input arguments is to provide values not existing in functions.

```
1   def get_revenue():  # Function without input
2       revenue = input("Enter the revenue:")
3       print("You have entered ", revenue)
4
5   def get_input(message): # Function with one input
6       user_input = input(message)
7       print("You have entered ", user_input)
8
9   def get_values(message, number): # Multiple inputs
10      for count in range(number):
11          user_input = input(message)
12          print("You have entered ", user_input)
```

NANYANG TECHNOLOGICAL UNIVERSITY | SINGAPORE



Using functions

- Use your own functions the same way as how you use other functions, call it by name.

```
1   def get_revenue():  # Function without input
2       revenue = input("Enter the revenue:")
3       print("You have entered ", revenue)
4
5   def get_input(message): # Function with one input
6       user_input = input(message)
7       print("You have entered ", user_input)
8
9   def get_values(message, number): # Multiple inputs
10      for count in range(number):
11          user_input = input(message)
12          print("You have entered ", user_input)
13
14
15  get_revenue()
16  get_input("Enter budget:")
17  num = 10
18  get_values("Enter data:", num)
```

8

NANYANG TECHNOLOGICAL UNIVERSITY | SINGAPORE

Allows for 2 variable to be included

Functions

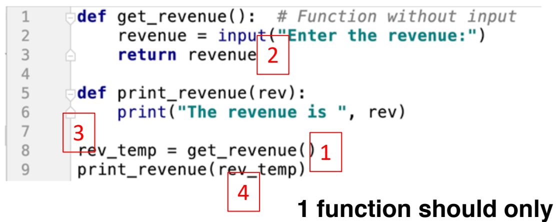
- Function is designed to encourage **code reuse.**
- One single function designed, can be **reused any number of times**.
- Function **organizes** and **hides the complexity of codes**.
- Caller only needs to call by name, fill in the **value(s)** to satisfy **input arguments**.

Variable scopes

- Variable scope can be local or global
  - Variable which can only be used or accessed within the function has local scope
  - Variable which can be used anywhere in the program has global scope
- Variables in function are meant to be used and discarded automatically immediately after. Those variables will not be accessible in any other function or main program.

Parsing of variables

- If a variable is needed by another function, it should parsed as return and input through main program.



```
def get_revenue():  # Function without input
    revenue = input("Enter the revenue:")
    return revenue

def print_revenue(rev):
    print("The revenue is ", rev)

rev_temp = get_revenue()
print_revenue(rev_temp)
```

1 function should only serve 1 purpose

Naming Convention

- Variable names, function names should be all lower case
  - Eg. budget, revenue
- If more than one word is required to describe the variable well, use **underscore**
  - Eg. return_on_investment or roi
- Names must be meaningful
  - Eg. ipc versus income_per_capita
- Capital letter is reserved for constant
  - Eg. PI = 3.1416

Program design

- Variable for storing value -> noun
  - Storing value that is likely to be used subsequently
- Function for performing a task -> verb
  - Put repetitive tasks to functions
- Flow of program is control by main function

Example of Program Design

Problem:

Write a program that asks the user to enter their name and their age. Your program will then compute the year user will turn 55. Your program will then print out a message addressed to them that tells user the year to withdraw their CPF savings (the year when they turn 55 years old).

Example of Program Design

Step 1:

Pick up **noun** and **tasks to do**.

- **Variable** for storing value -> **noun**
  - Storing value that is likely to be used subsequently
- **Function** for performing a task -> **verb**
  - Put repetitive tasks to functions

Example of Program Design

Step 1:

Pick up **noun** and **tasks to do**.

Write a program that asks the user to **enter** their **name** and their **age**. Your program will then **compute** the **year** user will turn 55. Your program will then **print out** a message addressed to them that tells user the year to withdraw their CPF savings (the year when they turn 55 years old).

Nouns: name, age, year
Tasks to do: enter, compute, print out

Example of Program Design

Step 2:
Define variables for **nouns**

Write a program that asks the user to **enter** their **name** and their **age**. Your program will then **compute** the **year** user will turn 55. Your program will then **print out** a message addressed to them that tells user the year to withdraw their CPF savings (the year when they turn 55 years old).

name = _______
age = _________
year = _________

NANYANG TECHNOLOGICAL UNIVERSITY | SINGAPORE

Example of Program Design

Step 3:
Call / make the relevant functions for **tasks to do.**

Write a program that asks the user to **enter** their **name** and their **age**. Your program will then **compute** the **year** user will turn 55. Your program will then **print out** a message addressed to them that tells user the year to withdraw their CPF savings (the year when they turn 55 years old).

name = _______
age = _________
year = _________

enter ⬄ input()
compute ⬄ + - * /
print out ⬄ print()

NANYANG TECHNOLOGICAL UNIVERSITY | SINGAPORE

Example of Program Design

Step 4:
Link up **variables** and **tasks to do.**

Write a program that asks the user to **enter** their **name** and their **age**. Your program will then **compute** the **year** user will turn 55. Your program will then **print out** a message addressed to them that tells user the year to withdraw their CPF savings (the year when they turn 55 years old).

```
name = input("Enter your name:")
age = int(input("Enter your age"))
year = 2020 - age + 55 # assuming current year is 2020
print("You will turn 55 in year ", year)
```

Example of Program Design

Step 5:
If this program is to be repeated -> make a new function (**calculate_cpf_yr**).

Write a program that asks the user to **enter** their **name** and their **age**. Your program will then **compute** the **year** user will turn 55. Your program will then **print out** a message addressed to them that tells user the year to withdraw their CPF savings (the year when they turn 55 years old).

```
def calculate_cpf_yr():
    name = input("Enter your name:")
    age = int(input("Enter your age"))
    year = 2020 - age + 55 # assuming current year is 2020
    print("You will turn 55 in year ", year)
```