Functions

Outline

- Functions
  - Define and call
  - Parameters and arguments
    - One arguments
    - Multiple arguments
  - Return
  - More on defining functions
    - Positional arguments and keyword arguments
    - Default arguments
    - Docstring

Types of functions

Type	Definition	Examples
Built-in Functions	Built-in functions are pre-defined functions that are provided by the programming language itself.	• `print()` • `len()` • `type()`
Methods	Methods are functions that are specifically defined within the context of certain data types.	• String methods: ▪ upper(), replace() • List methods: ▪ append(), remove()
User-defined function	A user-defined function is a function that you create yourself to perform a specific task.	`def func(x, y):` `  print (x + y )`

User-defined function

- Functions allow us to create blocks of code that can be easily executed many times, without rewriting the code.
- Later, if you make a change, you only need to make it in one place.

```
score = 7.5
if score > 7.0:
    print("pass")
else:
    print("fail")
```

```
pass
```

```
score = 6.0
if score > 7.0:
    print("pass")
else:
    print("fail")
```

```
fail
```

```
score = 8.0
if score > 7.0:
    print("pass")
else:
    print("fail")
```

```
pass
```

```
def decision(score):
    if score > 7.0:
        print("pass")
    else:
        print("fail")
```

```
decision(7.5)
```

```
pass
```

```
decision(6.0)
```

```
fail
```

```
decision(8.0)
```

```
pass
```

Define and call

- When you define a function, you use the keyword `def` and specify the name and the statements.

Function name

Parenthesis and colon

```
def greeting():
    print("Hi!")
    print("How are you?")
```

Statements

- Later, you can "call" this function by its name.

```
greeting()
```

```
Hi!
How are you?
```

Parameters and arguments

- Parameters are variables that will be used in the function.

parameter

```
def greeting(name):
    print("Hi, ",name)
    print("How are you?")
```

- Arguments are the actual values passed to the function when the function is called.

argument

```
greeting("Leo")
```

```
Hi, Leo
How are you?
```

```
greeting("Emma")
```

```
Hi, Emma
How are you?
```

same as

```
greeting(name = "Leo")
```

```
hi, Leo
How are you?
```

```
greeting(name = "Emma")
```

```
hi, Emma
How are you?
```

- Both parameters and arguments represent information used in the function.

One argument

- Example 1: Count the number of words in the text.

```
def word_counter(text):
    word_list = text.split()
    print("number of words: ",len(word_list))
```

```
word_counter('Python is a popular programming language.')
```

```
number of words:  6
```

```
word_counter("The Jupyter Notebook is a web-based interactive computing platform.")
```

```
number of words:  9
```

Pass a string argument to the function.

One argument

- Example 2: Write conditional statements in a function.

```
def grade(score):
    if score > 6:
        print ("pass")
    else:
        print ("fail")
```

```
grade(7)
```

```
pass
```

```
grade(5)
```

```
fail
```

Pass a number argument to the function.

One argument

- Example 3: Print item in list.

```
def shopping_cart(item_list):
    for i in item_list:
        print(i)
```

```
shopping_cart(["apple", "banana", "grape"])
```

```
apple
banana
grape
```

```
shopping_cart(["cheese", "yogurt", "milk"])
```

```
cheese
yogurt
milk
```

Pass a list argument to the function.

Exercise

Exercises.A

(A.1) Write a function that takes a parameter called `day` and prints out a greeting message based on the passed value. Test your function with (1) `day = "Monday"` (2) `day = "Friday"`.

Format:

Good morning! Today is **[day]**.

```
# define
```

```
# call
```

(A.2) Write a function that takes a parameter named `y` and prints the description of `y` based on its value. Test your function with (1) `y = 20` (2) `y = -15`.

test	print out
$y > 0$	positive
$y < 0$	negative
None of the above expression are true	zero

```
# define
```

```
# call
```

Multiple arguments

- Example 1: Add two numbers

```
def add_numbers(x, y):
    print(f"The sum of {x} and {y} is {x + y}.")
```

```
add_numbers(2,5)
add_numbers(6,3)
```

```
The sum of 2 and 5 is 7.
The sum of 6 and 3 is 9.
```

Multiple arguments

- Example 2: Print first n items from the list.

integer list

```
def first_n_items(n, items):
    for i in range(n):
        print(items[i])
```

```
n1 = 4
item_list1 = ["A", "B", "C", "D", "E", "F"]
first_n_items(n1, item_list1) # call function
```

```
A
B
C
D
```

```
n2 = 2
item_list2 = ["apple", "banana", "grape", "cheese", "yogurt"]
first_n_items(n2, item_list2) # call function
```

```
apple
banana
```

Exercise

Exercises.B

(B.1) Write a function that takes two parameters: `x` and `y`, and print out the product of `x` and `y`. Test your function with (1) x = 8, y = 2.5 (2) x = -7.5, y = 3.

```
# define
```

```
# call
```

```
# call
```

(B.2) Write a function that takes two parameters:

- `member` (bool): True if the customer is a member, otherwise, set it to False.
- `amount` (integer): The total amount of the order.

The function should calcuate and print the shipping fee according to the table below.

Shipping fee	Amount < 500	Amount >= 500
Non-members	69 kr	49 kr
Member	0 kr	0 kr

```
# define
```

```
# call
shipping_fee(member = True, amount = 350)
```

```
# call
shipping_fee(member = False, amount = 600)
```

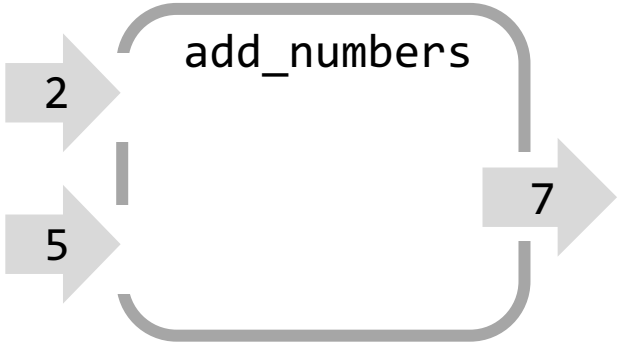
Return

- Use the keyword `return` to end the execution of the function call and return the result.

```
In [1]: def add_numbers(n1, n2):
            return n1 + n2

In [2]: add_numbers(2, 5)

Out[2]: 7
```



- The returned result can be assigned to a variable for further computation.

```
In [3]: x = add_numbers(2, 5)
        print (x)

        7
```

Store the returned value in a new variable named “x”

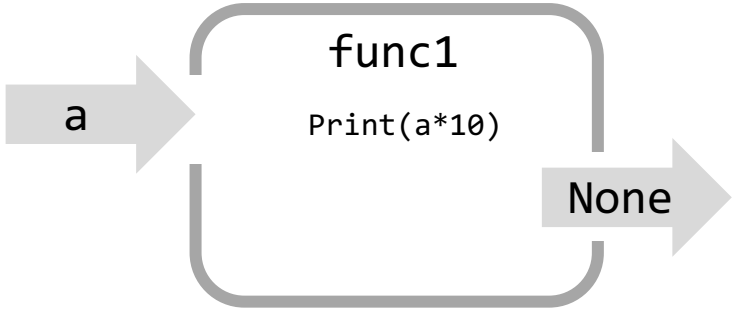
Return

- Compare functions without/with `return`

```
def func1(a):
    print(a*10)
```

```
func1(5)
```

```
50
```




```
x1 = func1(5)
```

```
50
```

The number "50" is printed in the function.

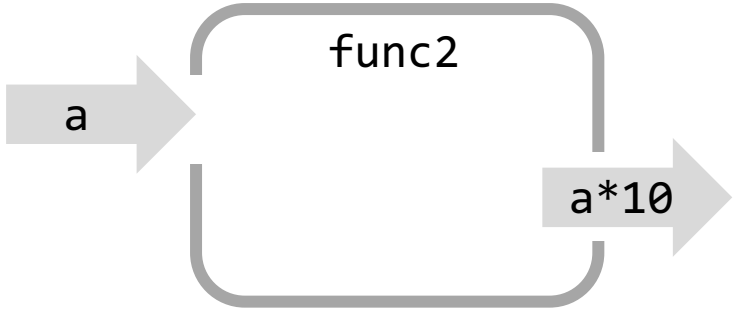
```
print(x1)
```

```
None
```

```
def func2(a):
    return a*10
```

```
func2(5)
```

```
50
```




```
x2 = func2(5)
```

The returned value " is assigned to the variable “x2”.

```
print(x2)
```

```
50
```

- **None** is not the same as 0, False, or an empty string. It is a data type of the class **NoneType** object.

Review

Build-in function	Example	Argument	Return
print	`print('hello world')`	`'hello world'`	`None`
type	`type(3.5)`	`3.5`	`float`
len	`len('hello world')`	`'hello world'`	`11`

Method	Example	Argument	Return
str.upper()	`s1 = 'hello'` `s1 .upper()`	`None`	`'HELLO'`
str.split()	`s2 = 'A-B-C'` `s2.split(sep = '-')`	`'-'`	`['A','B','C']`
list.append()	`numbers = [1, 2, 3]` `numbers.append(4)`	`4`	`None`

Return

- Return different values based on conditions.

```
def grade(score):
    if score > 6:
        return "pass"
    else:
        return "fail"
```

```
grade_student1 = grade(7)
grade_student2 = grade(5)
```

```
print(grade_student1)
print(grade_student2)
```

```
pass
fail
```

Return

- Call the function in a `for` loop and append all the returned values to the list.

```
def grade(score):
    if score > 6:
        return "pass"
    else:
        return "fail"
```

```
score_list = [7, 5, 8.5, 6, 7.5, 7, 5.5, 9, 8, 7.5]
grade_list = []
```

```
for s in score_list:
    g = grade(s)            #get grade by passing a score
    grade_list.append(g)    #append the grade to the list

print(grade_list)
```

```
['pass', 'fail', 'pass', 'pass', 'pass', 'pass', 'fail', 'pass', 'pass', 'pass']
```

Exercise

Exercise.C

(C.1) The table below shows the entrance fees to the Munch Museum for different age groups. Write a function that takes age as a parameter and returns the corresponding fare. Test your function with (1) `age = 23` **(2)** `age = 38` **.**

Group	Fare (NOK)
Adult	160
Young adult (age < 24)	100
Child (age < 16)	0

```
# define
```

```
# call
```

```
# call
```

(C.2) John wants to take his family to the Munch Museum. Use the function defined in (C.1) to get the fare for each family member. The list below contains the age of each family member.
Hint: Call the function in a for loop.

```
family = [41, 43, 5, 18, 13, 64]
```

```

```

Positional arguments and keyword arguments

```
def covid_stats(num_cases, country):
    print(f"There were {num_cases} confirmed cases in {country}.")
```

- Positional arguments are provided to a function based on the order in which the function's parameters are defined.

✓
```
covid_stats(315, "Norway")
```

```
There were 315 confirmed cases in Norway.
```

✗
```
covid_stats("Norway", 315)
```

```
There were Norway confirmed cases in 315.
```

- Keyword arguments are provided to a function by specifying the parameter names with their corresponding values.

✓
```
covid_stats(country = "Norway", num_cases = 315)
```

```
There were 315 confirmed cases in Norway.
```

By specifying the parameter names, you don't need to follow the order.

https://docs.python.org/3.11/tutorial/controlflow.html#keyword-arguments

Default arguments

- Default arguments are parameters in a function that have a predefined value.

```
def covid_stats(num_cases, country = "Norway"):
    print(f"There were {num_cases} confirmed cases in {country}.")
```

```
covid_stats(315)
```

```
There were 315 confirmed cases in Norway.
```

If you don't provide a value for “country”,
it uses the default value “Norway”.

- Non-default argument: num_cases
- Default argument: country

- When defining a function with both default and non-default arguments, the default arguments should come after the non-default arguments in the function.
- https://docs.python.org/3.11/tutorial/controlflow.html#default-argument-values

Docstring

- Python docstring provides a quick summary of a function.
- They typically include information about what the code does, how to use it, and any other relevant details.
- A docstring is declared using ’’’triple single quotes’’’ or “”””triple double quotes””” and should be written on the first line.

```python
def add_numbers(n1, n2):
    """
    This function takes two numbers as input and returns their sum.

    Parameters:
    n1 (int or float): The first number.
    n2 (int or float): The second number.

    Returns:
    int or float: The sum of n1 and n2.
    """
    return n1 + n2
```

```python
add_numbers?
```

```
Signature: add_numbers(n1, n2)
Docstring:
This function takes two numbers as input and returns their sum.

Parameters:
n1 (int or float): The first number.
n2 (int or float): The second number.

Returns:
int or float: The sum of n1 and n2.
File:      /var/folders/hx/z23rbxyx5xv63zy53xdl10qw0000gp/T/ipykernel_2810/2199623558.py
Type:      function
```

- “docstring” is short for "documentation string”.
- https://realpython.com/documenting-python-code/

Exercise

Exercise.D

(D.1) Given the following function. Call the function by first passing the arugment `name = "Maya"`, then the argument `city = "Bergen"`.

```
def greeting(city, name):
    print(f"Dear {name}, welcome to {city}.")
```

(D.2) Modify the function defined in (D.1): use "guest" as the default value for the paramenter "name". Test your function with (1) `city = "Oslo"` (2) `city = "Bergen", name = "Maya"`.