Functions

Outline

- Functions
  - Define and call
  - Parameters and arguments
  - Return
  - More on defining functions
- Exception handling

Functions

- 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.

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

```
pass
```

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

```
fail
```

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

```
pass
```

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

```python
decision(7.5)
```

```
pass
```

```python
decision(6.0)
```

```
fail
```

```python
decision(8.0)
```

```
pass
```

Build-in functions

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
`sum`	sum([1,2,3])	[1,2,3]	6
`str`	str(3.14159)	3.14159	“3.14159”

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.

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

argument

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

```
number_of_words('Apple, Alphabet and Microsoft rake in $57bn of quarterly profits.')
```

```
number of words:  10
```

```
number_of_words("Tesla solar and battery storage deployments tripled year-over-year in Q2 2021")
```

```
number of words:  11
```

argument

One argument

- Example 2: Write conditional statements in a function.

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

# call
grade(7)
grade(5)
```

```
pass
fail
```

One argument

- Example 3: Pass a list to a function.

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

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

```
apple
banana
grape
```

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

```
cheese
yogurt
milk
```

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("The sum is", x+y)
```

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

```
The sum is 7
The sum is 9
```

Multiple arguments

- Example 2: Search words that start with a given letter.

List string

```
def search_word(word_list, letter):
    for word in word_list:
        if word[0] == letter:
            print(word)
```

```
friend_list = ["Henry", "Victoria","Isaac", "Sara", "Zoe","Isabelle", "Nora", "Madelyn", "Sophia",
             "Charlotte", "Michael","Sebastian","Leah", "Ryan", 'Matthew', "Mila"]
```

```
search_word(friend_list, "S")
```

```
Sara
Sophia
Sebastian
```

```
search_word(friend_list, "I")
```

```
Isaac
Isabelle
```

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
```

(B.2) Write a function named `find_max` that takes two lists as arguments. Use the following format to print out the maximum value of each list.
Format:
The maximum value in the first list is `_`, and the maximum value in the second list is `_`.

```
#define
```

```
#call
find_max([1,2,3],[6,7,8])
```

```
#call
find_max([10,50,20,40],[25,60,40,35,65,10])
```

Return

- Use the keyword `return` to <u>end the execution</u> of the function call and <u>return the result</u>.

```
def add_numbers(num1, num2):
    return num1+num2
```

```
x = add_numbers(2,5)
print (x)
```

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

```
7
```

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

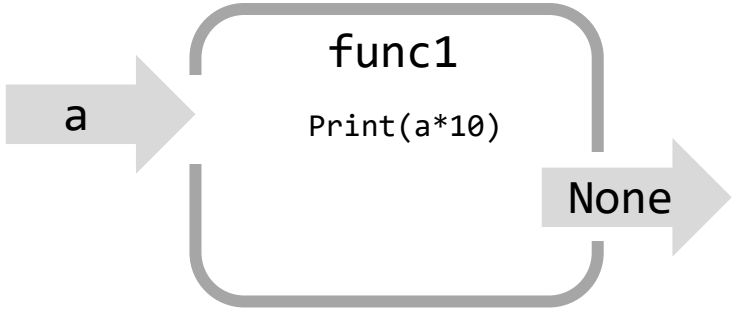
Return

- 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.

Return

- Return a list

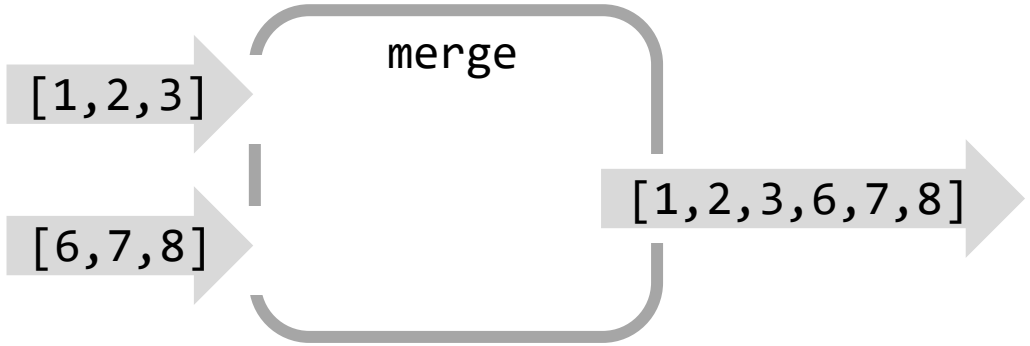
```
def merge(list_a, list_b):
    return list_a + list_b
```

```
list_new = merge([1,2,3],[6,7,8])
```

Store the returned list in a new variable named “list_new”

```
print(list_new)
```

```
[1, 2, 3, 6, 7, 8]
```



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]
```

```

```

More on defining functions

- You can specify the parameter names with values so that you do not need to follow the order of the parameters.

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

✓

```
# Call the function by passing two arguments
covid_stats(315, "Norway")
```

Positional arguments

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

✗

```
# If you pass the arguments in the wrong order
covid_stats("Norway", 315)
```

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

✓

```
# By specifying the argument name, you don't need to follow the order
covid_stats(country = "Norway", num_cases = 315)
```

Keyword arguments

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

https://docs.python.org/3.8/tutorial/controlflow.html#keyword-arguments

More on defining functions

- You can provide a default value to a parameter by using the assignment operator (=).

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

```
# call the function without passing the argument "country"
covid_stats(315)
```

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

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

- Non-default argument should not follow the default argument.
- https://docs.python.org/3.8/tutorial/controlflow.html#default-argument-values

More on defining functions

- Python docstring provides a quick summary of a function.
- A docstring is declared using ’’’triple single quotes’’’ or “”””triple double quotes””” and should be written on the first line.

```
In [138]: def test(name):
              '''This is a test function'''
              print("Hi", name)

In [139]: test?
```

```
Signature: test(name)
Docstring: This is a test function
```

https://docs.python.org/3.8/tutorial/controlflow.html#documentation-strings

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"`.

Exception handling – Try Except

- Use `try` and `except` to respond to the occurrence of an exception.

```
x = "10"        # x is incorrectly defined
print(x/2)      # cause a TypeError exception
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_22424/4237868325.py in <module>
      1 x = "10"        # x is incorrectly defined
----> 2 print(x/2)      # cause a TypeError exception

TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

```
x = "10"
try:
    print(x/2)
except TypeError:
    print("The data type of x is incorrect.")
```

```
The data type of x is incorrect.
```

https://docs.python.org/3.8/tutorial/errors.html#handling-exceptions

Exception handling – Try Except

- Catch NameException

```
try:
    print(z/2)
except NameError:
    print("Variable z is not defined.")
```

```
Variable z is not defined.
```

- Other Exceptions

```
try:
    print(x[3])
except:
    print("Something went wrong")
```

```
Something went wrong
```