Functions

Outline

- Functions
  - Define and call
  - 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.

```
In [1]: age = 10
        if(age>=67):
            print("Senior (100 NOK)")
        elif(age<=15):
            print("Children (Free)")
        else:
            print("Adults (140 NOK)")

        Children (Free)

In [2]: age = 24
        if(age>=67):
            print("Senior (100 NOK)")
        elif(age<=15):
            print("Children (Free)")
        else:
            print("Adults (140 NOK)")

        Adults (140 NOK)

In [3]: age = 45
        if(age>=67):
            print("Senior (100 NOK)")
        elif(age<=15):
            print("Children (Free)")
        else:
            print("Adults (140 NOK)")

        Adults (140 NOK)
```

```
def get_price(age):
    if(age>=67):
        return "Senior (100 NOK)"
    elif(age<=15):
        return "Children (Free)"
    else:
        return "Adults (140 NOK)"
```

```
get_price(10)
```

```
'Children (Free)'
```

```
get_price(24)
```

```
'Adults (140 NOK)'
```

```
get_price(45)
```

```
'Adults (140 NOK)'
```

Build-in functions

function	example	argument	return
`print`	print(“hello world”)	“hello world”	hello world
`type`	type(3.5)	3.5	float
`len`	len(“hello world”)	“hello world”	11
`float`	float(5)	5	5.0
`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?
```

Arguments

- Information can be passed to the function as arguments.
- Arguments are specified in parentheses after the function name.

argument

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

- When the function is called, we pass a name.

```
greeting("Lucas")
```

```
Hi, Lucas
How are you?
```

```
greeting("Maya")
```

```
Hi, Maya
How are you?
```

Arguments

- Example

```
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
```

Arguments

- Write conditional statements in functions

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

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

```
pass
fail
```

Exercise

(A.1) Define a function named `warning`. Inside the function, print out the message "This website uses cookies to imporve user experience." Call this function.

Expected result:

```
This website uses cookies to imporve user experience.
```

(A.2) Write a function with an argument y. Print out the description of y. Test your function by using (1) $y = 20$ (2) $y = -15$.

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

Arguments

- Multiple arguments

```
def add_numbers(num1, num2):
    print("The sum is",num1+num2)
```

```
x=2
y=5
add_numbers(x,y)
x=3
y=6
add_numbers(x,y)
```

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

Arguments

- Multiple arguments

```
def compare_numbers(a, b):
    if a > b:
        print(a, 'is greater than',b)
    elif a < b:
        print(a, 'is less than', b)
    else:
        print('Two numbers are equal')
```

```
compare_numbers(10, 20)
```

```
10 is less than 20
```

```
compare_numbers(100, 50)
```

```
100 is greater than 50
```

Arguments

- Multiple arguments
  - Example: 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

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

(B.2) Given the following list, write a function named `get_products` with two arguments `product_list` and `category`. Print out matching products according to the specified category.

Example:

category = "meat"

Expected reuslt:

```
['meat', 'beef']
['meat', 'pork']
```

```
#[product category, product name]
product_list = [["beverage","coffee"],["dairy","cheeses"],["meat","beef"],["meat","pork"],["beverage", "tea"],["dairy","yogurt"],
product_list
```

```
[['beverage', 'coffee'],
 ['dairy', 'cheeses'],
 ['meat', 'beef'],
 ['meat', 'pork'],
 ['beverage', 'tea'],
 ['dairy', 'yogurt'],
 ['beverage', 'soda'],
 ['dairy', 'milk']]
```

```
# test your function
get_product(product_list, "meat")
```

Product category	Product name
beverage	coffee
dairy	cheeses
meat	beef
meat	pork
beverage	tea
dairy	yogurt
beverage	soda
dairy	milk

Return

- If we do not want to print values directly, we can use the keyword `return` to store them in new variables.

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

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

Store the returned value in a new variable

```
7
```

Return

- Return a list

```
def search_word(word_list, letter):
    matched_words = []
    for word in word_list:
        if word[0] == letter:
            matched_words.append(word)
    return matched_words
```

```
list_S = search_word(friend_list, "S")
list_I = search_word(friend_list, "I")
print(list_S)
print(list_I)
```

Store the returned list in a new variable

```
['Sara', 'Sophia', 'Sebastian']
['Isaac', 'Isabelle']
```

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

(C.1) Write a function named `remainder`, with two arguments, `a` and `b`. Return the remainder of a÷b. Test your function by passing (1) a = 29, b = 5 (2) a = 16, b = 3. Print out the returned values.

Expected result:

```
4
1
```

(C.2) Write a function with two arguments, `mylist` and `n`. Return the average of first `n` values in `mylist`. Test your function by passing (1) mylist = [2,10,9,5,11,24,6,17], n = 3 (2) mylist = [2,10,9,5,11,24,6,17], n = 5. Print out the returned values.

Expected result:

```
7
7.4
```

More on defining functions

- You can specify the argument name with values so that you do not need to remember the order of arguments.

```
def covid_stats(num_cases, country):
    print("There were {} confirmed cases in {}.".format(num_cases, 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 an argument by using the assignment operator (=).

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

(D.1) Call the function defined in (C.1) and specify the argument name: b = 5, a = 29.

```
```

(D.2) Rewrite the function defined in (C.2) by using 3 as the default value of the argument `n`. Call the function by passing `mylist`.

```
```

Type of errors

- When an error occurs, Python will stop and generate an error message.
- Syntax errors

```
print "hello world"
```

```
  File "<ipython-input-2-6d29d8fb337c>", line 1
    print "hello world"
          ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print("hello world")?
```

- Build-in exceptions
  - e.g., TypeError, NameError, ZeroDivisionError

```
#Type error
str1 = "50"
str1/2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-12-94baa66c9975> in <module>
      1 #Type error
      2 str1 = "50"
----> 3 str1/2

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

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)
<ipython-input-13-647877f95ba1> 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("Please enter a valid number.")
```

```
Please enter a valid number!
```

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
```