Exercises 2: Python programming

Introduction to data science and Python programming for medical research (2025)

1 Exception handling

In the last exercise from yesterday the script would crash if the user would give inputs for the blood pressure that could not be cast to a number. Copy your code from that solution into a new Python script file.

- Put the part with the user input and cast to `float` in a `try-except` block.
- Print any error messages in the `except` part of the block.
- Add a Boolean variable before the `try` block which should indicate if the user input was successful. The value should change inside the `try-except` block. You need to consider what value it should have before, `True` or `False`, and what it should have after. There are more than one way to solve this problem.
- Use the Boolean variable to only execute the blood pressure classification part of the code if user input was successful. Use an additional level of `if-else` block.

2 Containers: Lists

- Initialize a list that should contain four different numbers, ints, floats or a mix. Print the list.
- Use `len(...)` to get and print the size of the list.
- Use append to add an additional two numbers. Print the list and its length.
- Add an `input` statement that should prompt the user for an index, *i.e.* and integer. Use the index and `[]` brackets to print the value in the list at that index.
- `sort` the list with the build in sorting function in the list.
- Use slicing to create a variable containing the sub list of values at index 1 and 2 in the list. Print the original list and the new sub list.

3 Containers: Dictionaries

- Create an empty dictionary using `{}`.
- Add four key-value pairs
  1. `'firstname'`, a string (value also in all lowercase).
  2. `'lastname'`, a string (value also in all lowercase).
  3. `'department'`, a string (value also in all lowercase).
  4. `'year-phd-student'`, an integer value.
- Print the dictionary.
- Use key indexing and string formatting to print:
  `<firstname> <lastname>: <department> (<year>)`
  e.g. `Nils Olovsson: IGP (5)`
- Use the string functions `capitalize` and `upper` to capitalize the first and last name and write the department in all capital letters. Print in a similarly formatted string as the one before.

4 Control flow (for-loop)

Copy the code for creating the list from exercise 2.

- Iterate over all values in the list using a `for` loop and print the individual values.
- Add a second for loop and iterate over all values in the list using a `range` and the length of the list. Print both the index and the value in the list at the index for all values in the list.
- Add a third `for` loop where you iterate over the list using enumerate to get both the index and the value. Print both.

5 Dictionary iterations

Copy the code for creating the dictionary from exercise 3.
Use `items()` to iterate over all key-value pairs in the dictionary. Print both.

6 Functions

Write a function that computes the length of the hypotenuse of a right angled triangle using the formula

$$c = \sqrt{a^2 + b^2}$$

- The function should take two arguments, `a` and `b`.
- You can look at the lecture notes from yesterday to find the operator for computing powers of numbers.
- You need to `import` the standard `math` library to access the square root method `math.sqrt(...)`.
- The function should return the value `c`.
- Call the function y adding code below the function definitions in the same file.
- Add a suitable, short, docstring and type hint annotations.

7 Modules

Create a new file in the same directory as your code for these exercises called `MyMath.py`.

- Copy the code for the function from exercise 6 into the new file.
- Add a second function that computes the area of a disk according to the formula

  $$A = \pi r^2$$

  where r is the radius and π is the mathematical constant which can be found in the `math` standard package.
- `import MyMath` as a module in a separate file and call both of your functions using the parameter values

  1. $a = 3$ and $b = 5$
  2. $r = 8.2$

  Print the results.

8 Classes

- Download or copy the file `Animal.py` into your working directory.
- Extend the `Animal` class by writing your own class that inherits from `Animal`. The new class implement overrides for `make_sound(self)` and `get_number_of_legs(self)`.
- Use the string conversion that is already present in Animal to print an instance of your new class.

```python
from abc import ABC, abstractmethod
class Animal(ABC):
    def __init__(
        self,
        nr_of_legs: int,
    ):
        self.nr_of_legs = nr_of_legs
        return

    def __str__(self) -> str:
        name = self.__class__.__name__
        msg = f'{name} has '
        msg += f'{self.nr_of_legs} '
        msg += 'legs and says '
        msg += f'"{self.get_sound()}"'
        return msg

    @abstractmethod
    def get_sound(self) -> str:
        pass

class Spider(Animal):
    def __init__(self):
        Animal.__init__(self, 8)
        return

    def get_sound(self) -> str:
        return 'imse vimse'

class Cow(Animal):
    def __init__(self):
        Animal.__init__(self, 4)
        return

    def get_sound(self) -> str:
        return 'Mooo'

# Create instances of animals
spider = Spider()
cow = Cow()
print(spider)
print(cow)
```

9 File and time

- Open a file in read state and print it line by line. You can use `end=''` to prevent double line breaks in each print.
- Add two `time.time()` statements, one before and one after the file reading.
- Calculate and print the time it took to open and print the content of the file.

- The time is measured in seconds but the file reading was probably much faster than a second. Convert the time to a more suitable unit, such as μs and print the time indicating this unit.

10 Optional: Code analysis

Use `flake8` and `mypy` to analyze the style of your code in exercises 7 and 8.