Week 2: Functions

Contents

Important

You must start by downloading the script

> download_week02.py

Right-click and select "Save link as...". You may need to confirm the download in your browser.

Run this script in `VSCode` to download the files you need for week 2. The script does not need to be located a specific place on your computer.

Exercise 2.1: Full name

Complete the function `full_name` in file `cp/ex02/full_name.py` which takes as input two strings `first_name` and `last_name`. The function should print a full name consisting of `first_name` and `last_name` with exactly one space character in between.

```
>>> full_name('Vedrana', 'Dahl')
Vedrana Dahl
>>> full_name('Morten', 'Hannemose')
Morten Hannemose
>>> full_name('Donald', 'Duck')
Donald Duck
```

`cp.ex02.full_name.full_name(first_name, last_name)`

Determine and print the full name given first name and last name.

Parameters:

- **first_name** (`str`) – your first name.
- **last_name** (`str`) – your last name.

Skip to main content

Exercise 2.2: Next thousand

Complete the function `next_thousand` in file `cp/ex02/next_thousand.next_thousand.py` which takes as input a single number `a`. The function should print the value of `a` rounded to the next thousand. For example:

```
>>> next_thousand(123998)
124000
>>> next_thousand(-123998)
-123000
```

cp.ex02.next_thousand.next_thousand(*a*)

Round a number to the next nearest thousand and print.

Parameters:

a (`int`) – the number to be rounded.

Exercise 2.3: Name length

Complete the function `name_length` in file `cp/ex02/name_length.py` which takes as input a string `name`. The function should print a message informing about the length of `name`. For example, if `name='Anika'` the function should print `"Your name consists of 5 characters."`

```
>>> name_length('Anika')
Your name consists of 5 characters.
```

cp.ex02.name_length.name_length(*name*)

Calculate and print the number of characters in name.

Parameters:

name (`str`) – your name.

Tip

Exercise 2.4: Wind chill

Complete the function `wind_chill` in file `cp/ex02/wind_chill.py` that takes as input the actual temperature `temperature` and the wind speed `windspeed`. The function should print a message informing about the actual and the perceived temperature, with all numbers rounded to the nearest integer (for example, `'Temperature: 17 degrees feels like 15 degrees.'`). The perceived temperature is denoted by wind chill temperature T_{WC}. The cooling effect depends on air temperature T and speed v.

$$T_{\mathrm{WC}} = 13.12 + 0.6215T - 11.37v^{0.16} + 0.3965Tv^{0.16}$$

```
>>> wind_chill(8, 12.8)
Temperature: 8 degrees feels like 6 degrees.
>>> wind_chill(8, 25.8)
Temperature: 8 degrees feels like 4 degrees.
>>> wind_chill(-2, 12.8)
Temperature: -2 degrees feels like -6 degrees.
```

Skip to main content

`cp.ex02.wind_chill.wind_chill(temperature, windspeed)`

Calculate and print the wind chill temperatures.

Parameters:

- **temperature** (`int`) – the actual temperature.
- **windspeed** (`float`) – the wind speed.

Exercise 2.5: Normal weight

The body mass index (BMI) is defined as:

$$\text{BMI} = \frac{w}{h^2}$$

where w is body weight (mass) in kilograms, and h is the height in metres. A person is categorized as being of normal weight if the BMI falls in range from 18.5 to 25.

Implement the function `normal_weight` in file `cp/ex02/normal_weight.py` that takes height `height` in meters as input. The function should print a message about normal weight range, given as the smallest and the largest weight in whole kilograms, which when converted to BMI fall into normal range. The message should say, for example, `'Normal weight is between 56 and 74 kg.'`

Example: for a given height h = 1.73, the limits of the normal weight range can be calculated as:

$$\begin{aligned} w_{low} &= 18.5 \cdot 1.73^2 \\ &= 55.3687 \\ w_{high} &= 25 \cdot 1.73^2 \\ &= 74.8225 \end{aligned}$$

We need weight in whole kilograms and to ensure that weight falls into the normal range, the lower limit should be rounded up and the upper limit should be rounded down. The function should print the message `'Normal weight is between 56 and 74 kg.'`

```
>>> normal_weight(1.73)
Normal weight is between 56 and 74 kg.
```

`cp.ex02.normal_weight.normal_weight(height)`

Calculate and print the range of normal weights for a given height.

Parameters:

height (`float`) – the height.

Exercise 2.6: Survival temperature

To survive in cold temperatures, human must either maintain a sufficiently high metabolic rate, or regulate heat loss by covering their body with clothes made of insulating material. Given the metabolic heat production M and thermal conductance (for clothes and tissue) g, the lowest temperature for survival T can be calculated by:

Skip to main content

$$T = 36 - (0.9M - 12) \cdot \frac{g + 0.95}{27.8g}$$

The reasonable range for metabolic heat production is from 50 for sleep to 500 for heavy activity, e.g. sports. The reasonable range for thermal conductance is from 0.04 for winter sleeping bag to 0.45 for no clothing.

Implement a function `survival_temperature` that takes the metabolic heat production M `metabolic_heat` and the thermal conductance g `thermal_conductance` as input. The function should print a message about the survival temperature. For example, `'Survival temperature is -2.8 degrees.'`. The temperature should be rounded to the nearest 1 decimal place.

```
>>> survival_temperature(200, 0.1)
Survival temperature is -27.5 degrees.
```

cp.ex02.survival_temperature.survival_temperature(*metabolic_heat*, *thermal_conductance*)

Calculate and print the lowest survival temperature.

Parameters:

- **metabolic_heat** (`int`) – the metabolic heat production.
- **thermal_conductance** (`float`) – the thermal conductance.

Exercise 2.7: Unit conversion

Implement a function `unit_conversion` in file `cp/ex02/unit_conversion.py` that takes a length in foot and inch as input. The function should print a message about the corresponding length in centimeter. For example, `'7 ft 5 in is equal to 226 cm.'`. The length should be rounded to the nearest integer.

```
>>> unit_conversion(7, 5)
7 ft 5 in is equal to 226 cm.
```

cp.ex02.unit_conversion.unit_conversion(*foot*, *inch*)

Convert length in foot and inch to centimeter.

Parameters:

- **foot** (`int`) – foot portion of the length in imperical unit.
- **inch** (`int`) – inch portion of the length in imperical unit.

Exercise 2.8: Hadlock's formula for fetal growth

In obstetric ultrasound, fetal weight is widely used to monitor growth of the baby. This can be estimated from the ultrasound image by using the Hadlock's formula:

$$\log_{10} \mathrm{EFW} = 1.326 + 0.0107 \cdot \mathrm{HC} + 0.0438 \cdot \mathrm{AC} + 0.158 \cdot \mathrm{FL} - 0.00326 \cdot \mathrm{AC} \cdot \mathrm{FL}$$

where EFW is the estimated fetal weight (g), HC is the head circumference (cm), AC is the abdominal circumference (cm), and FL is the femur length (cm).

Implement a function `hadlock` in file `cp/ex02/hadlock.py` that takes the three measurements hc `head_circ`, ac `abdomial_circ`, and fl `femur_length` as input. The function should print a message about the estimated fetal weight. For example, `'The estimated fetal weight is 2306.7 g.'`. The estimated fetal weight should be rounded to the nearest 1

Skip to main content

```
>>> hadlock(31.1, 30.2, 8.3)
The estimated fetal weight is 2990.7 g.
```

`cp.ex02.hadlock.hadlock(head_circ, abdominal_circ, femur_length)`

Estimate fetal weight using Hadlock formula.

Parameters:

- **head_circ** (`float`) – head circumference in cm.
- **abdominal_circ** (`float`) – abdominal circumference in cm.
- **femur_length** (`float`) – femur length in cm.

Exercise 2.9 (optional): Keyword arguments and default values

So far you've been writing Python functions that takes in one or more input arguments, perform a few computational steps, and print a message containing the results. In this exercise, we'll explore how Python functions can be used in few other ways.

When multiple values are passed as input arguments to a function, it can be difficult to remember the order in which the arguments should be passed.

```
>>> def keyword_arguments_demo(arg1, arg2):
...     print("arg1 is:", arg1, ", arg2 is: ", arg2)
...
>>> keyword_arguments_demo(7, 3)
arg1 is: 7 , arg2 is:  3
>>> keyword_arguments_demo(3, 7)
arg1 is: 3 , arg2 is:  7
```

However, we can specify the arguments by their names instead, which makes the code easier to read, especially when the variables have self explanatory names. We can give the arguments in an arbitrary order, as long as we specify the name of the argument.

```
>>> def keyword_arguments_demo(arg1, arg2):
...     print("arg1 is:", arg1, ", arg2 is: ", arg2)
...
>>> keyword_arguments_demo(arg1=7, arg2=3)
arg1 is: 7 , arg2 is:  3
>>> keyword_arguments_demo(arg2=3, arg1=7)
arg1 is: 7 , arg2 is:  3
```

In many cases, it is useful to give default values to some of the inputs in the function definition. This allows us to call the function without specifying all the arguments:

```
>>> def keyword_arguments_demo(arg1=12, arg2=5):
...     print("arg1 is:", arg1, ", arg2 is: ", arg2)
...
>>> keyword_arguments_demo(25, 7)
arg1 is: 25 , arg2 is:  7
>>> keyword_arguments_demo(25)
arg1 is: 25 , arg2 is:  5
>>> keyword_arguments_demo(arg2=7)
arg1 is: 12 , arg2 is:  7
```

Notice how the denominator has a default value of 3 when it's not passed as input arguments to the function.

Skip to main content

Your exercise is to try this out with your own functions!