# Exercises

## Set 3

DM857 Introduction to Programming
DS830 Introduction to Programming

## 1 Handling Errors

1. For each error type below, write a python program that exhibit such error without using `raise`.

   (a) `ValueError` Raised when an operation or function receives an argument that has the right type but an inappropriate value, and the situation is not described by a more precise exception.

   (b) `TypeError` Raised when an operation or function is applied to an object of inappropriate type. The associated value is a string giving details about the type mismatch.

   (c) `SyntaxError` Raised when the parser encounters a syntax error.

   (d) `NameError` Raised when a local or global name is not found. This applies only to unqualified names.

   (e) `AssertionError` Raised when an assert statement fails.

2. For each of the following programs, find all the possible sources of errors. Discuss which should or can be addressed using preconditions, `try/except`, or other changes to the program.

   (a)
   ```
   a = input('Enter a value for a:')
   b = input('Enter a value for b:')
   c = a * int(b)
   ```

   (b)
   ```
   a = input('Enter a value for a:')
   b = input('Enter a value for b:')
   c = a / int(b)
   ```

   (c)
   ```
   x = 5
   s = 'x is '
   print(s + x)
   ```

   (d)
   ```
   def quota(jobs,workers):
       return jobs / workers
   ```

   (e)
   ```
   def fahrenheit_to_celsius(degrees):
       return degrees * conversion_factor
   ```

   (f)
   ```
   def get_int(message):
       s = input(message)
       i = int(s)
       return i
   ```

3. For each of the following programs, show its output and reconstruct how errors propagate.

   (a)
   ```
   try:
     x = int(input('x = '))
   except ValueError:
     x = 0
   finally:
     print(x)
   ```

   (b)
   ```
   try:
     y = 1 / int(input('y = '))
   except ValueError:
     y = 1
   except ZeroDivisionError:
     y = 0
   finally:
     print(y)
   ```

   (c)
   ```
   z = 0
   try:
     z = 1 / int(input('z = '))
   except ValueError:
     z = 1 / z
   except ZeroDivisionError:
   ```

```
    z = 2
  finally:
    print(z)
```

(d)
```
try:
    x = 0
    y = 1 / x
    print(y)
  except ZeroDivisionError:
    x = 1
  finally:
    print(x)
    print(y)
```

(e)
```
x = 1
try:
  try:
    y = int(input('y = '))
  except ValueError:
    x = 0
  finally:
    print(1 / x)
except ZeroDivisionError:
  x = 2
finally:
  print(x)
```

(f)
```
try:
  try:
    z = int(input('z = ')) / 0
  except ValueError:
    z = 0
  finally:
    print(1 / z)
except ZeroDivisionError:
  print(z)
```

## 2 Reading inputs from the keyboard

1. Write a function `input_yes_no(prompt:str)->bool` that queries the user for a yes/no answer by displaying the message `prompt`.
2. Write a function `input_pos_int(prompt:str)->int` that queries the user for a positive integer displaying the message `prompt`.
3. Write a function `input_float(prompt:str,min:float,max:float)->int` that queries the user for a floating point number in the range `min`,...,`max`.
4. Write a function `input_int(prompt:str,min:int,max:int)->int` that queries the user for an integer in the range `min`,...,`max`.
5. Change your function `input_float` to make the lower (`min`) and upper (`max`) bound for range of accepted inputs optional (in a call we might provide a value for both, either, or neither `min` or `max`). For instance, a call

   ```
   >>> input_float('Please enter a non-positive number: ', max=0)
   ```

   accepts any floating point number smaller or equal to $0$; and a call

   ```
   >>> input_float('Please enter a probability: ', min=0, max=1)
   ```

   accepts any number between $0$ and $1$ (included).

   (Hint: type `float` has values that represent positive and negative infinities and module `math` defines `inf` as the value for positive).
6. Change your function `input_int` to make the lower (`min`) and upper (`max`) bound for range of accepted inputs optional. For instance, a call

   ```
   >>> input_int('Please enter a natural number: ', min=0)
   ```

accepts any integer greater or equal to $0$.

(Hint: make `min` and `max` of type `int | None`. Module `typing` defines a shorthand for types that express an optional argument like this one called `Optional`.[1] Specifically, `Optional[int]` is `int | None`.)

7. Write a function `input_choice(prompt:str,choice1:str,choice2:str)->str` that queries the user to select between `option1` and `option2` and returns the user's selection. For instance:

```
>>> input_choice('Where should we go?','left','right')
Where should we go?
 1. left
 2. right
2
'right'
```

8. Write program `coin.py` where the users has to guess if a (virtual) coin flip will yield heads or tails. After each toss, the program prompts the user with the choice to quit or play again. To virtually flip a coin, you can use the function `randint(a,b)` from module `random`: this function returns a random integer between `a` and `b` (included).[2] Below is an example of a possible interaction on the console.

```
Please, pick 'h' for heads or 't' for tails: f
I am sorry, 'f' is not a valid input.
Please, pick 'h' for heads or 't' for tails: t
The result is heads, too bad.
Would you like to play again? (yes/no)
no
Ok, bye!
```

9. Write a program `areas.py` for computing the area of circles, rectangles, squares, and triangles. The program has a main menu where the user can select one of the supported shapes or quit the program. If the user picks a shape, then the program asks the necessary lengths, prints the result, and resumes the main menu. If the user chooses to quit, then the program terminates.

```
Enter
  1 to compute the area of a circle
  2 to compute the area of a rectangle
  3 to compute the area of a square
  4 to compute the area of a triangle
  0 to quit this program
2
Enter the width of the rectangle: 4.0
Enter the height of the rectangle: 2.0
The area is 8.0.
```

[1] https://docs.python.org/3/library/typing.html#typing.Optional
[2] https://docs.python.org/3/library/random.html#random.randint

# 3 Phone Book Manager

In this exercise you will develop a program for managing a phone book. Numbers follow the 8-digit format with optional spaces separating pairs of digits e.g.,`'12345678'`, `'1234 5678'`, `'12 34 56 78'`. The program consists of two modules:

- `phone_book` which is responsible for maintaining the phone book in memory and offers functions for searching and updating entries of the phone book (see description below).
- `manager` which is the main module of program and is responsible for handling the interaction with the user (you will implement this module, instructions are at the end).

## 3.1 Module `phone_book`

The module is implemented by `phone_book.py` available on itsLearning and provides the functions described below.

- `is_present(name:str)->bool` checks if there is an entry for the given name.
- `get(name:str)->Optional[str]` returns the phone number stored under the given name or `None` if there is no entry for `name`.
- `add_or_update(name,phone)->None` adds an entry to the phone book or updates its information if there is already an entry for the given name. The function raises `ValueError` if the string `phone` is not a phone number (8-digit format).
- `delete(name)->None` remove the entry for the given name (assumes the entry is present).

## 3.2 Module `manager`

Your task will be to write `manager` (using `phone_book`). The program needs to present the user with an interactive prompt where the user can select one of the following commands:

- `search` to search the phone book by name.
- `set` to create or edit an entry regardless of whether it is already in the phone book.
- `add` or `new` to create a new entry.
- `edit` or `update` to edit an existing entry.
- `del` or `delete` to delete an entry.
- help to display the list of available commands.
- `quit` to terminate the program (see function `exit` of module `sys`).

Then, the program performs the corresponding action and, if the action is not termination, awaits a new command.

You must follow a top-down approach. We can organise the program functionality in three layers:

1. The top layer handles the main interaction where the user enter a command. This layer will call functions from the next layer.

2. The second layer handles each specific operation (searching, etc.). This layer will require module `phone_book` and the third layer.

3. The third layer consists of a function for handling the input of phone numbers.

When implementing a layer, you should write "mock" implementations of the lower layers i.e. functions with the expected signature that act as placeholder and do not fully implement all the necessary functionality. This will allow you to test your program as you proceed through the layers. For instance, while implementing the first layer, you will need to rely function provided by the second layer, e.g., a function `search` that handles the implements the necessary interactions with the user and module `phone_book` to search for a contact. This is basic mock for `search`:

```
def search()->None:
    """ Interacts with the user to search a contact by name. """
    # TODO: implement input and search
    print("Search has not been implemented yet.")
```

A more sophisticated mock of the function could actually input the name but forego searching the phone book, for instance:

```
def search()->None:
    """ Interacts with the user to search a contact by name. """
    # TODO: implement search
    name = input("Enter the name to search: ")
    print("Search has not been implemented yet.")
    print(f"You entered '{name}'.")
```