Exercises

Set 1

DM857 Introduction to Programming
DS830 Introduction to Programming

1 Arithmetic expressions, types, and variables

1. For each of the following expressions write the order in which it is evaluated.

   (a) `a + b * c ** d`
   (b) `a + b ** c * d`
   (c) `a / b // c * d`
   (d) `a + b - c - d`
   (e) `a ** b - c ** d`
   (f) `a ** b ** c - d`
   (g) `a / b - c * d`
   (h) `a * b ** c * d`
   (i) `a % b / c ** d`
   (j) `a - b - - c - - - d`

2. Consider the following assignments

```
>>> i = 3
>>> f = 2.19
```

   Find the type (int or float) for each of the following expressions.

   (a) `i + 3 * i`
   (b) `(i + 3.0) * i`
   (c) `1 - i + 2`
   (d) `3.0 + i * i`
   (e) `9 ** 0.5`
   (f) `i ** 2 // 2`
   (g) `i ** 2 // f`
   (h) `i ** f // i`
   (i) `i / i - 2`
   (j) `i / f * f`

   Then, check your solutions with IDLE/Shell.

3. Remove unnecessary parenthesis for each of the following expressions (`a`,`b`,`c`, and `d` are variables).

   (a) `a + ((b * c) - d)`
   (b) `(a * b) / (c * d)`
   (c) `(a + b) + (c + d)`
   (d) `a ** (b / (c ** d))`
   (e) `(a ** -(b)) ** (c ** d)`

4. For each of the following statements find if it results in a `SyntaxError`, a `NameError`, or neither assuming you just started IDLE/Shell (each is the first statement you entered).

(a) `>>> x = 6`

(b) `>>> x = y`

(c) `>>> y = y + 5`

(d) `>>> 5 = x`

(e) `>>> z -= 6`

(f) `>>> z =- 6`

(g) `>>> z -=- 6`

(h) `>>> z += 6`

(i) `>>> z =+ 6`

Then, check your solution with IDLE (you may need to clear previous assignments: click on "Restart Shell" under the menu "Shell").

5. For each of the following code snippets, find the value associated with each variable at the end of the execution. Draw a sketch of each of them in the Memory Model representation.

(a)
```
>>> i = 1
>>> j = 2
>>> j = 3 + i * 2
>>> i = j / 2 * i + 3
```
(b)
```
>>> i = 3
>>> j = 3.0
>>> j = j - 2.3
>>> i = i + j
```
(c)
```
>>> i = 3
>>> j = 0
>>> i /= 2
>>> j += 0.0
>>> i = i % 2
```
(d)
```
>>> i = 3
>>> j = 3
>>> i = i ** j
>>> j = i ** 0.5
>>> j = j // i
```

Then, check your solution with IDLE.

6. Write a program `seconds.py` that converts a time interval given in hours, minutes, and seconds into the equivalent time interval in seconds. The input should be stored in variables at the beginning of the program (later we will learn how to ask users for inputs) and the output should be printed on screen when the program is run. Remember to structure and comment your code. Test your program with different input values.

7. Write a program `durations.py` that converts a time interval given in seconds into one in a readable format (hours, minutes, and seconds). The input should be stored in a variable at the beginning of the program and the output should be printed on screen when the program is run. Think about the algorithm to use and which extra variables are useful. Remember to structure and comment your code. Test your program with different input values.

8. Write a program `formula.py` that evaluates the mathematical formula $\sqrt{\frac{|x^2-y^2|}{2}}$ given values for x and y and prints the result. (recall that $\sqrt{z}$ is $z^{\frac{1}{2}}$). Remember to structure and comment your code. Test your program with $x = 2$ and $y = 6$ and check that the result is 4, then test it with different input values.

2 Programming with functions

1. Show how Python executes each of the following code snippets.

(a)
```
>>> def double(x : float) -> float:
...     ''' doubles the number x '''
...     return x * 2.0
>>> x = 2
```

(b)
```
>>> def double(x : float) -> float:
...   ''' doubles the number x '''
...   return x * 2.0
>>> x = double(2)
```

(c)
```
>>> def double(x : float) -> float:
...   ''' doubles the number x '''
...   return x * 2.0
>>> x = double(2) + double(5) ** 2
```

2. For each of the following code snippets, identify the scope of each name and find their associated value at the end of the execution.

(a)
```
>>> def f(x):
...   return x + 1
>>> x = f(2)
```

(b)
```
>>> def f(f):
...   return f + 1
>>> x = f(2)
```

(c)
```
>>> def f(x):
...   return x + s
>>> s = 2
>>> x = f(2)
```

(d)
```
>>> x = 1.1
>>> def f(y):
...   z = x + y
...   return x
>>> z = f(2)
```

(e)
```
>>> def f(x):
...   return x+1
>>> def f(x):
...   return x-1
>>> x = f(0)
```

(f)
```
>>> def f(g,x):
...   return g(x)
>>> x = f(abs,-1.1)
>>> y = f(round,-1.1)
```

(g)
```
>>> def f():
...   return g(abs, 2)
>>> def g(g,x):
...   return g(x - 10)
>>> x = f()
```

Warning: The snippets above are meant to illustrate tricky aspects of name scopes in Python. They are by no means examples of good code writing, especially snippets b, e, and g which contain **extremely bad** naming practices.

3. For each function in the snippets (2.a–2.e) above, propose type hints that are consistent with the function use.

4. Define

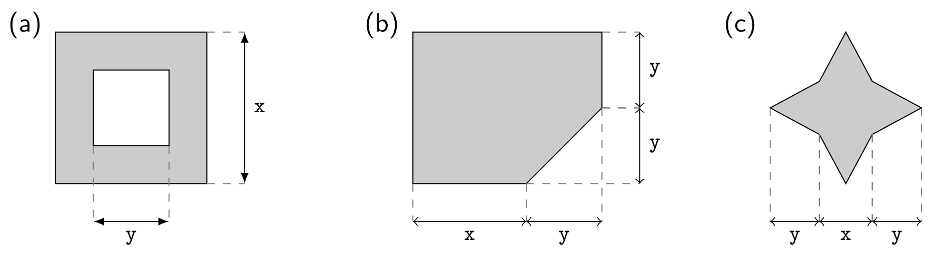
   (a) a function `rectangle_area(width,length)` that returns the area of a rectangle given its width and length.

   (b) a function `square_area(side)` that returns the area of a square given its side.

   (c) a function `triangle_area(base,height)` that returns the area of a triangle given its base and height.

   Remeber to document your definitions by writing appropriate type hints and docstrings (including examples for testing).

5. For each of the following figures, compute the greyed area using the functions defined in the previous exercise.



3 Writing, documenting, and testing functions

1. A developer working on a program `bill_plz.py` to help people split bills wrote the following functions. .

```python
def round_up(x):
    """ Rounds up a floating number."""
    # integer division by 1 rounds down: 1.1 // 1 is 1.0, -1.1 // 1 is -2.0
    # so we negate, divide, and negate again to round up before truncating
    # the result to an integer.
    return int(-(-x // 1))

def bill_quota(bill, tip, people):
    """
    Returns the amount each person needs to contribute to cover a bill and
    the corresponding tipping percentage (0.2 is a 20% tip).
    """
    total = bill + bill * tip
    quota = round_up(total / people)
    return quota
```

   The documentation of both functions is incomplete and their contract is unclear. They might also contain bugs.

   - Write appropriate type hints, tests, and document preconditions (if any).
   - Using the Shell, check that the function `help` prints the expected message when called on one of your functions (e.g., `help(round_up)`).
   - Check that both functions behave as promised and that all tests are cleared.
   - Should you find bugs, fix them.

   (You can have python run all the tests in the functions docstring checking the results for you using `doctest` e.g., by launching `python -m doctest check_plz.py` from the command line.)

2. In a file `quadrilaterals.py` write functions to compute perimeter and area of squares, rectangles, rhombuses, parallelograms, kites, darts, isosceles trapezia and right trapezia. Remember to document each function contract. Write and run test to check your implementation. You might find helpful defining auxiliary functions for computing triangle areas and hypotenuses (prefix the name of your auxiliary functions with one underscore e.g., `_hypotenuse`).