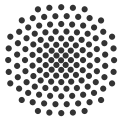

University of Stuttgart

Programming Paradigms Prof. Dr. Michael Pradel

University of Stuttgart, Summer 2023

Exercise 3: Control flow

(Deadline for uploading solutions: June 8, 2023, 11:59pm Stuttgart time)

The materials provided for this homework are:

- a pdf file with the text of the homework (this);
- a zip file with **the folder structure and the templates that must be used for the submission**.

The folder structure is shown in Figure 1.

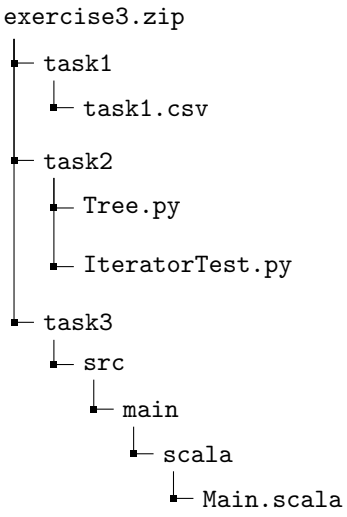

Figure 1: Folder structure to use for submission.

The submission must be compressed in a zip file using the given folder structure. The names of folders and files must not be changed or moved, otherwise the homework will not be evaluated.

General tips for making sure your submission is graded as you expect:

- Use only the zip format for the archive (**NOT** rar, 7z, gz, or anything else).
- Do not rename files or folders, simply open the files provided and put your solutions.

1 Task I (25% of total points of the exercise)

The goal of this task is to evaluate the given expressions under different sets of rules for precedence and associativity. Table 1 defines two sets of rules.
Task: (To submit your answer, please fill in the file *exercise3/task1/task1.csv*.)

- Using the Rules 1 in Table 1, evaluate the following expressions:
  1. `5 ** 2 >> 6 / 3`
  2. `2 ** 2 ** 3 / 4 + 3 * 2`
  3. `12 - 2 * 4 != 1 && 9 + 10 / 2 > 12`
- Using the Rules 2 in Table 1, evaluate the following expressions:
  4. `3 * 2 + 4 << 3 << 1`
  5. `2 * 4 + 9 < 20 || 45 / 5 + 4 != 5`
  6. `5 * 3 ** 2 + 5 > 6 ** 3 >> 4 / 2`

Operator	Rules 1		Rules 2		Type	
	Assoc.	Prec.	Assoc.	Prec.	Operands	Results
**	R	2	R	2	numeric	numeric
*	L	3	R	3	numeric	numeric
/	L	3	R	3	numeric	numeric
+	L	4	R	3	numeric	numeric
-	L	4	R	3	numeric	numeric
>>	L	5	R	4	numeric	numeric
<<	L	5	R	4	numeric	numeric
>	L	6	L	5	numeric	boolean
<	L	6	L	5	numeric	boolean
!=	L	7	L	6	numeric	boolean
&&	L	8	L	7	boolean	boolean
||	L	8	L	7	boolean	boolean

Table 1: Two sets of rules for precedence and associativity. A lower number in the "Prec." column means higher precedence. The "Assoc." column indicates whether the operator is left-associative ("L") or right-associative ("R").

Note: The meaning of individual operators is the same as in Java. Additionally, '`**`' represents the exponential (power) operation (e.g., `2 ** 3` is `8`). To indicate the result of evaluating an expression, use the syntax specified by Java literals (e.g., the number five is `5` and the Boolean values are `true` and `false`). Furthermore, all literals are integer and as such consider integer division.

Evaluation Criteria: Your solution will be compared against the correct result of evaluating each expression under each set of rules.

2 Task 2 (20% of total points of the exercise)

This task is to implement a "true" iterator, i.e., a generator, in Python. You are given a Python class that implements a tree, where each node stores a list of strings. Your task is to implement the `iterate_strings_in_all_nodes()` method of the `Tree` class to iterate through all strings in all nodes of the tree. The order of iteration is up to you.

Note: The assignment should be implemented in Python (version >= 3.8). Submissions that do not comply with the exact folder structure and names in Figure 1, or that do not work with Python (version >= 3.8), cannot be graded.

Evaluation Criteria: Your code will be evaluated by checking the logic and results. We will run your Python script in a fresh and new Python environment that contains only standard Python packages. We do not permit the installation of new packages, therefore make sure you do not use any dependencies that are not included in the standard Python library.

3 Task 3 (55% of total points of the exercise)

For this task, you have to implement three tail-recursive functions in Scala:

1. `filterGreaterThanBase`, which checks whether the integers in an array are greater than the base integer.
2. `isDaffodil`, which checks whether an integer is a narcissistic integer.
3. `countLetters`, which counts how many times each letter occurs in a string.

The assignment should be implemented in Scala (version = 3.2.2). You can get Scala at `https://www.scala-lang.org/download/`. You must implement the functions in such a way that they are tail-recursive, as described in the lecture. As a brief recap, a function is tail-recursive when it calls itself as its last instruction, so that the current stack frame can be reused for the next call.
Writing a tail-recursive function helps the compiler performing some optimizations, and Scala has a special annotation `@tailrec` to tell the compiler that a function should be treated as a tail-recursive function. This feature helps you detect errors early, since a function that is annotated as tail-recursive but cannot be recognized as such by the compiler will raise a compile-time error.

3.1 Starting Point

You will find the template Scala project to use for this task in the */task_3* folder. The 'src' folder contains the only source files you must use for this task. Your solution must be entirely implemented following this template, using the source file (`Main.scala`) and no other files. Please keep the names of the three functions called in the source code unchanged. You are free to use the IDE of your choice.

Your task is to implement the function body of the three functions in `Main.scala`. You are allowed to use only Scala standard libraries.

3.2 Testing

Examples of correct input-output pairs for the three functions:

Input	**Output**
`filterGreaterThanBase(Array(5, 21, 17, 4, 10, 15), 12, Array.empty[Int])`	`21, 17, 15`
`isDaffodil(200, 0)`	`false`
`isDaffodil(153, 0)`	`true`
`countLetters("aadabcbbbcc", Map.empty[Char, Int])`	`Map(a -> 3, d -> 1, b -> 4, c -> 3)`

Use these input-output pairs to check whether your functions work as expected. We will use additional tests to evaluate your solution, and you are advised to also use further tests for your own testing.

3.3 Evaluation Criteria

Your code will be evaluated against a held-out set of test cases that exercise your code. Submissions that do not comply with the exact folder structure and names in Figure 1, or that do not work with Scala (version = 3.2.2), cannot be graded.