

## Programming Summer 2020

### Exercises

Number 02, Submission Deadline: May 2, 2020

1. **String formatting.** Often, computational results are reported in form of text, where several pieces of information are composed into a single sentence, e.g.: “The sum of 4 + 10 + 28 is 42”. Python provides a convenient way of constructing such strings through the use of *place holders*, as shown here by two examples: (6 P)

```
a = 4
b = 10
c = 28
# first example
my_string = 'The sum of {} + {} + {} is {}'.format(a, b, c, a + b + c)
print(my_string)
# second example (notice the leading "f" in front of the string!)
my_string2 = f'The sum of {a} + {b} + {c} is {a + b + c}'
print(my_string2)
```

Read the “Guide to the Newer Python String Format Techniques” at <https://realpython.com/python-formatted-output/> to inform yourself about the `format()` function and f-strings.

- (a) Find the formatting instruction (using the `format()` function) that produced the following textual output for the numbers 12, 2947, and 60947.65<sup>1</sup>:

```
□□□□12
□2,947
60,948
```

Make sure to use the *same formatting instruction* to print the requested text for each of the numbers.

- (b) Explain in detail the formatting instructions that have been used in the following statement:

```
'{{ {1:2f}}-{{0:010.2f}}:{{1:b}} }'.format(1234.5678, 23)
```

- (c) Provide a meaningful output formatting for the following list of books using f-strings:

```
books = [
    {'title': 'To Kill a Mockingbird', 'author': 'Harper Lee',
     'isbn': 9780062420701, 'price': 12.99},
    {'title': 'Pride and Prejudice', 'author': 'Jane Austen',
     'isbn': 9781909621657, 'price': 7.19},
    {'title': '1984', 'author': 'George Orwell',
     'isbn': 9781328869333, 'price': 10}]
```

2. **Length function.** Python has a *builtin*<sup>2</sup> function called `len()` through which the *length* of an instance of a data type can be computed, e.g. `len(['this list has one element'])` returns 1. Which of the data types that you learned in the lecture are valid input of the function? (2 P)

<sup>1</sup>white spaces (□) are only visualized for your convenience

<sup>2</sup>“builtin” means that this function is provided *per se*

3. In the lecture, you got a very brief introduction into Python's slice notation for ordered collections and strings. For example, `my_list[:3]` will return the first three elements of the list `my_list`. Inform yourself about the capabilities of the slice notation to answer the following questions: (2 P)

- (a) How to extract the last three elements of a list?
- (b) How to extract all elements of odd positions of a list?

4. **Set.** Which data types can be stored in a `set`? (1 P)

5. **Implicit Boolean conversion.** In Python, the conversion of non-Boolean data types in Boolean expressions is *implicit*, as illustrated in the following: (3 P)

- `False` or `'This is a text'` evaluates to `'This is a text'`,
- `12` and `13` evaluates to `13`,
- `0` or (`None` and `'This is a text'` and `False`) evaluates to `None`

To understand this behavior of Python, remember that Python evaluates statements from left to right. Also, Python makes use of lazy-evaluation, i.e., it stops the evaluation of the expression as early as its result becomes obvious. For instance, in the third example, the expression `'This is a text' and False` is not evaluated, because `None` already falsified the `and` conjunctions.

Evaluate the following Boolean expression and explain your result. Specify the position at which Python stops the evaluation:

- (a) `0 and 'This is a text' or ''`
- (b) `age = 15.5`  
`age > 16 and 'You can buy beer' or 'No alcoholic ' + \`  
`'beverages for minors, sorry'`
- (c) `('a' and 0) or (False or (-1 and 4 > 10))`

6. **Elif clauses.** Next to `if` and `if-else` clauses, Python also allows `if-{elif}*` and `if-{elif}* -else` clauses, where the expression `{elif}*` means that the “`elif`” statement can be repeated an arbitrary number of times. The `elif` clause allows to make case distinctions such as the one shown in the following example: (1 P)

```
a = 'Jane'
if a == 'Mary':
    print('Gotcha! I knew it was you, Mary')
elif a == 'John':
    print('John! What a surprise!')
elif a == 'Jane':
    print('Of all people, I expected you the least, Jane!')
else:
    print('Sorry, but I\'m lost. Who are you?')
```

Use the `if-{elif}* -else` clause to check the type a given variable `a`. Similar to the example above, do four case distinctions to check three types of your choice. Use the `print` function to reveal the variable's type in a full sentence.

### Important:

Please submit your solution as (adequately commented) Python file. Use the cell separator comment `"#%"` to partition your Python file analog to the six exercises. Make sure your Python file contains only valid Python code.