Variables, Console I/O, Functions & Branching

Ahmet Uysal
auysal16@ku.edu.tr

Date: 5th of February

1 Overview

This handout is prepared for KOLT Python Certificate Program. It contains a brief review of this week's topics and exercise questions.
You can download the starter code from here.
You can find the solution here after all the sections are conducted.

2 Review

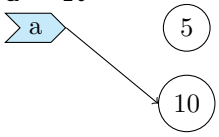
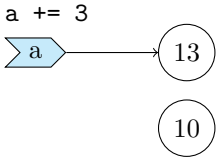
2.1 Variables

Variables are how we represent & store data in Python. You can imagine them as **labels** pointing towards data (Python objects).

2.1.1 Objects

Everything is an object in Python. Even though variables **do not** have `types`, each object has a **fixed** `type`. Values at the right side of our label analogy are objects!

2.1.2 Example

`a = 5` 	This is an assignment operation. Python first evaluates the **right-hand** side and creates an **integer object** to represent this value. After right part is done, Python points the label `a` to this object.
`a = 10` 	This is also an assignment operation. Python creates the object and points the label `a` to it. One question is what happens to old integer object? Python automaticly deletes object without any `references`(labels pointing to them).
`a += 3` 	`a += 3` is equivalent to `a = a + 3`. Therefore, this is also an assignment operation. You know the rest of the story.
`print(a)`	This is a **function call**. When a function has called, Python evaluates its **parameters**. The variable `a` is pointing to integer object 13. `print` function prints `13` to console.

2.2 Object Types

Type	Explanation	Examples
`int`	represent **integers**	3, 4, 17, -10
`float`	represent **real numbers**	3.0, 1.11, -109.123123
`bool`	represent **boolean** truth values	`True`, `False`
`str`	A sequence of characters.	'Hello', '', '3'
`NoneType`	special and has one value, None	`None`

You can learn the `type` of an object by calling the `type` function.

2.3 Console I/O

`print(*args, sep=' ', end='\n')`

- Can take arbitrary number of arguments
- Separates elements with single space by default
- Adds newline character `'\n'` to end by default

`input([prompt])`

- Prints the prompt to Console
- Program is paused until user enters something
- **returns an `str` object!**

2.4 Simple Functions

Functions are blocks of **organized**, **reusable** code that carry some **specific** tasks. We will talk more about functions in more detail soon, however, we wanted to briefly introduce the most simple form of functions to help you write better Python code. In Python, indentation is used to group different expressions. Recommended amount for an indentation level is 4 spaces.

```
def function_name():
    <expression>
    <expression>
    <expression>
    ...
```

Code snippet 1: Syntax for defining a simple function

2.5 Branching

We might want to control how our program behaves based on different conditions. ¡*condition*¿ can be any Python expression, Python will evaluate this expression and decide on which `code block` will be executed based on whether this expression has a **falsy** or **truthy** value.

```
if <condition>:
    <expression>
    <expression>
    ...
```

Code snippet 2: `if` statement, code block will be executed only if the condition is **truthy**.

```
if <condition>:
    <expression>
    <expression>
    ...
else:
    <expression>
    <expression>
    ...
```

Code snippet 3: `if-else` statement. If condition is a **truthy** value, first code block (if block) is executed. Otherwise, the second block (else block) is executed.

```
if <condition>:
    <expression>
    <expression>
    ...
elif <condition>:
    <expression>
    <expression>
    ...
...
else:
    <expression>
    <expression>
    ...
```

Code snippet 4: `if-elif-else` statement. You can add as many `elif` blocks as you want. Think about the difference between having `elif` blocks and `if` blocks.

```
# 'Falsy' values
bool(None) # => False
bool(False) # => False
bool(0) # => False
bool(0.0) # => False
bool('') # => False
# Empty data structures
bool([]) # => False
# Everything else is 'truthy'
bool(-100000) # => True
bool('False') # => True
bool(3.14) # => True
bool(int) # => True
# Nonempty data structures
bool([1, 'a', []]) # => True
bool([False]) # => True
```

Code snippet 5: Truthy and Falsy values.

3 Exercises

3.1 FizzBuzz

FizzBuzz is a group game that is usually played to teach division to children. In computer science, it is also a popular interview question. We will implement a slightly modified version of it.

Your program should take a integer from the user, then it should print:

- an error message 'You entered a negative number!'if the number is negative,
- Fizz, if this number is a multiple of 3
- Buzz, if this number is a multiple of 5
- FizzBuzz, if this number is a multiple of 15
- The number itself otherwise

3.2 Leap Year

Roman general Julius Caesar introduced the first leap years over 2000 years ago. But the Julian calendar had only one rule: any year evenly divisible by four would be a leap year. This formula produced way too many leap years. Still, it was not corrected until the introduction of the Gregorian calendar more than 1500 years later.

In the Gregorian calendar, three criteria must be taken into account to identify leap years:

- The year can be evenly divided by 4;
- If the year can be evenly divided by 100, it is NOT a leap year, unless;
- The year is also evenly divisible by 400. Then it is a leap year.

Write a program that takes a year from the user and prints whether it is a leap year or not in the Gregorian calendar.

3.3 Birthday

Did you ever wondered on which day of the year you are born? Our team member Necla wants the learn the answer of this question, but she is too busy praparing for the next lecture. Help Necla satisfy her curiosity.

Write a program that first takes the year, month (as integer), and day of the user's birth date from one by one. Then, calculate and print on which day of the year the user has been born.

References

[1] Fizz buzz. (2020). Retrieved 4 February 2020, from https://en.wikipedia.org/wiki/Fizz_buzz

[2] Leap Day on February 29. (2020). Retrieved 4 February 2020, from https://www.timeanddate.com/date/leapyear.html