Functions

Ayşe Turşucular
atursucular18@ku.edu.tr

Date: 26th 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 solutions at here after all the sections are conducted.

2 Review

2.1 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

2.2 Branching

We might want to control how our program behaves based on different conditions. A *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>
    ...

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

Code snippet 2: `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.

2.3 While loops

We can utilize `while` loops when we want to repeat some `<expression>`s as long as a `<condition>` is `True`.

```
while <condition>:
    <expression>
    <expression>
    ...
```

Code snippet 3: `while` statement, code block will be executed only if the condition is **truthy**. After each execution of code block, Python jumps back to condition check stage.

2.4 Strings

Textual data in Python is handled with `str` objects, or strings. Strings are immutable sequences of Unicode code points. String literals are written in a variety of ways:

- **Single quotes**: 'allows embedded "double" quotes'
- **Double quotes**: "allows embedded 'single' quotes".
- **Triple quoted**: '''Three single quotes''', """Three double quotes"""

Triple quoted strings may span multiple lines all associated white space will be included in the string literal.

2.4.1 Indexing

Strings are collections of characters. We can access specific characters using **indexing**. Let's look at an example:

```
my_name = 'hasan'
```

How can we access **third** character of this string?

```
   0  1  2  3  4
'  h  a  s  a  n  '
  -5 -4 -3 -2 -1
```

Notice that **indexing starts at zero** instead of one. Therefore, we need to use `my_name[2]` instead of `my_name[3]`. Alternatively, we can use negative indices.

2.4.2 String Operations

`str1 + str2` ⇒ Concatenate `str1` and `str2`
`str1 * `n ⇒ Repeate `str1` n times.

```python
print('This a simple calculator program.')
number1 = input('Please enter the first number:')
number2 = input('Please enter the second number:')
print(f'{number1}+{number2} is {number1 + number2}')

number3 = int(input('Third number:'))
number4 = input('Please enter the fourth number:')
print(f'{number3}x{number4} is {number3 * number4}')
```

Code snippet 4: String operation examples.

2.5 Functions

Functions are blocks of organized, reusable code that carry some specific tasks.

```python
def function_name():
    <expression>
    ...
def function_name(parameter1, parameter2, ...):
    <expression>
    ...
```

Code snippet 5: Defining a function only makes it available. You should call the function to execute.

- *return* statement immediately terminates the function.

```python
def function_name(parameter1, parameter2, ...):
    <expression>
    ...
    return value
```

Code snippet 6: Functions implicitly return None if they complete without a return statement.

- The values of parameters can be set to as default.

```python
def person_info(name, surname, age):
   print(name, surname, age)
```

```python
#Specific values
person_info('Ayse', 'Tursucular', '20')

#It prints;
Ayse Tursucular 20
```

Code snippet 7: When you sign a value to the parameter, it will execute.

```python
#This function returns the absolute value of the entered number
def absolute_value(number):
  if number >= 0:
    return number
  else:
    return -number

#For num = 48
new_num=absolute_value(48)
print(new_num)
#It prints 48

#For num = -78
print(absolute_value(-78))
#It prints 78
```

Code snippet 8: An example for function of return statement.

3 Exercises

3.1 Perfect Number

Implement a program which find all the perfect numbers between 1 and 10000.

- A perfect number is a positive integer which is equal to the sum of its proper positive divisors.

1. Create a boolean to decide if number is perfect or not.
2. Find the sum of proper divisors of the given number excluding itself.
3. Print the perfect numbers in a given range.

```
This program finds the perfect numbers between 1 and 10000.
1
6
28
496
8128
```

3.2 Hangman

Remember the old times that you were playing hangman with your friends? You are asked to implement that nostalgic game. Player has a predetermined number of lives and the game terminates if the player runs out of lives or correctly guesses the whole word.

1. Pick a random word from a list of predefined options.
2. Take the guess letter from player and continue to ask for a new letter if player enters invalid letter. (lenght!=1)
3. Implement the game!

```
Welcome to Hangman! You have 6 lives to correctly guess the word.
______
Your guess: P
P_____
Your guess: Y
PY____
Your guess: T
PYT___
Your guess: H
PYTH__
Your guess: N
PYTH_N
Your guess: O
Congratulations, you won!
```

3.3 Math Quiz

Ahmet is struggling with mathematical operations and he wants to practice them. You are asked to implement a program for Ahmet that asks user to answer randomly generated math questions.

1. First, pick the operation randomly. It can be *summation*, *subtraction* or *multiplication*.
2. Then generate two random numbers between (0,100) (must be changed every time user enters a correct answer)
3. Compare the answer of the user and the correct answer. If user's answer is true, count and print it when user does not give the correct answer.
4. Program should be running until user's answer does not match with the correct answer.

```
This is a math quiz program, GOOD LUCK!
It will continue until user enters a wrong answer
54 * 96 = 5184
20 + 60 = 80
81 * 53 = 2
Number of correct answers:  2
```

References