While Loops, Turtle Graphics & Strings

Ahmet Uysal
auysal16@ku.edu.tr
Hasan Can Aslan
haslan16@ku.edu.tr

Date: 12th 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>
    <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.

2.3 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"""

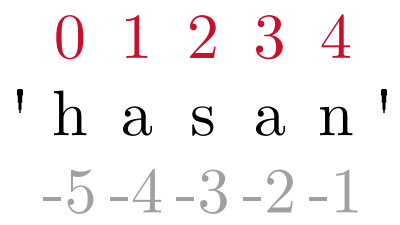
Triple quoted strings may span multiple lines all associated whitespace will be included in the string literal.

2.3.1 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?



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.

2.3.2 Slicing

We can **slice** strings by using `[start:stop:step]`

```
s = 'Python'
s[1] # => 'y'
s[0:4] # => 'Pyth'
s[:3] # => 'Pyt'
s[3:] # => 'hon'
s[:] # => 'Python'
s[:5:2] # => 'Pto'
s[1:4:3] # => 'y'
s[::3] # => 'Ph'
s[::-1] # => 'nohtyP'
```

Code snippet 5: **Indexing** and **slicing** examples.

2.3.3 String Operations

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

```
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 6: String operation examples.

2.4 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 7: `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.

2.5 `turtle` module

We will use Python's builtin `turtle` module to implement some visual examples using the consepts we have learned so far. To get more information, you can check out *the official Python documentation of `turtle` module* or you can refer to *Python Turtle Graphics Slides from the University of Auckland*.

3 Exercises

3.1 Rock Paper Scissors

Bool is bored, let's play rock paper scissors with her. The rules are basic

- Rock covers paper
- Scissors cut paper
- Rock breaks scissors

In order to make the game more fun, Bool is playing against the Computer Cool, so your chances are random. We have implemented the randomness, and as a hint we advice you to use two `bool` values named win and tie to check the result of the round and print the results afterwards.

3.2 Circles

We will use the `turtle` module and our programming skills to create a design. Your task is to get how many circles(n) this design should consist of from the user, and then construct this design.
Hint: You may want to use `turtle.circle(RADIUS)` and `turtle.left(angle)` functions.

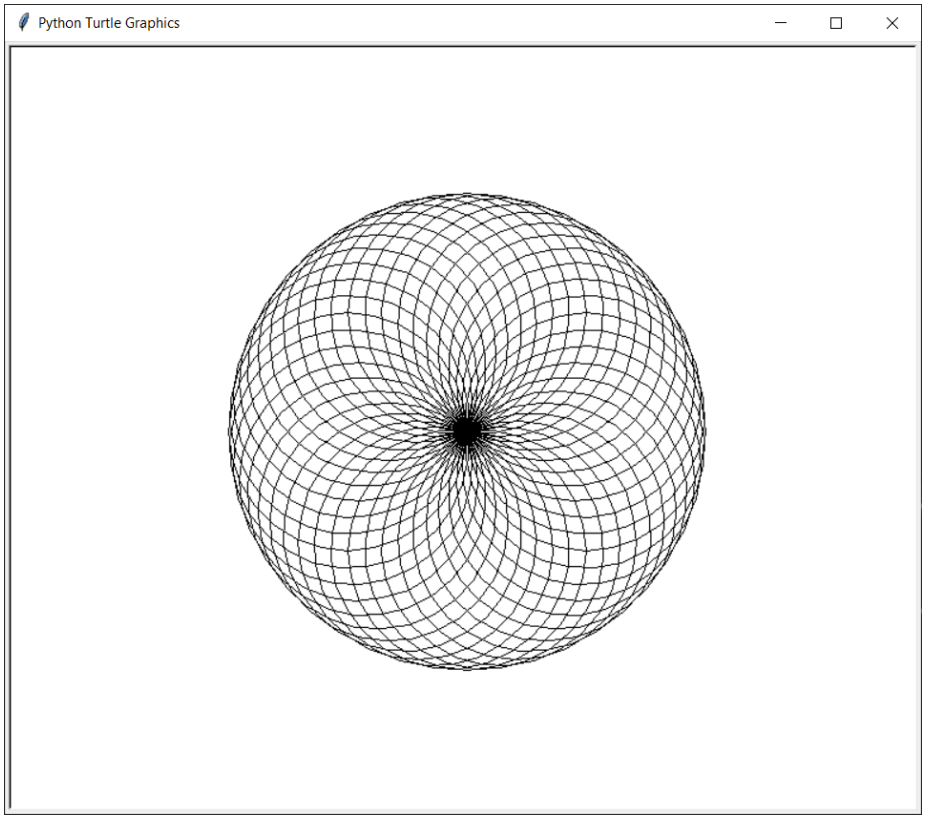


Figure 1: Sample output for $n = 44$.

3.3 Stars

Now, we will use our skills to create an even more complex shape: a star with an arbitrary number of edges.
According to our calculations, we can accomplish this goal by repeating a simple process n times where n is the number of edges.

1. Draw a line
2. Rotate θ angles (counterclockwise)

$$\text{Where } \theta = \begin{cases} 180 - (360/n) & \text{if } n \text{ is even} \\ 180 - (180/n) & \text{if } n \text{ is odd} \end{cases}$$

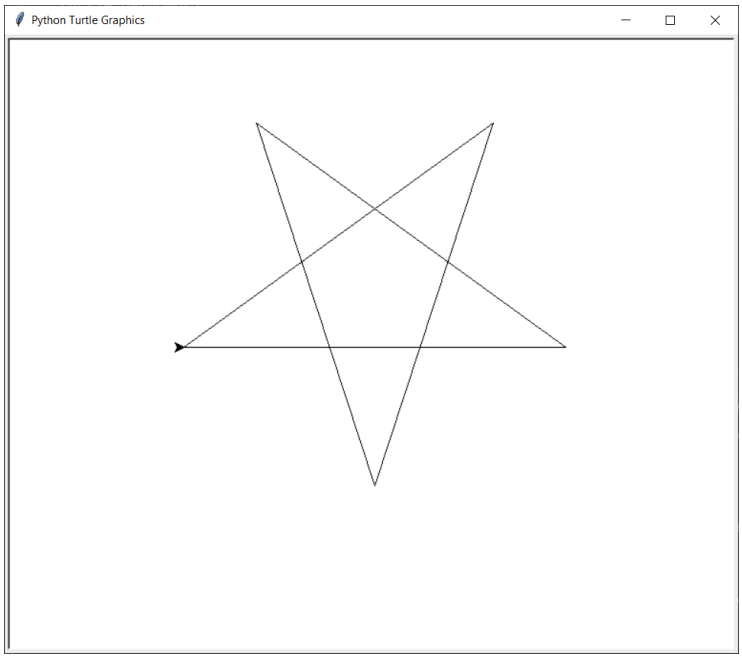


Figure 2: Sample output for $n = 5$.

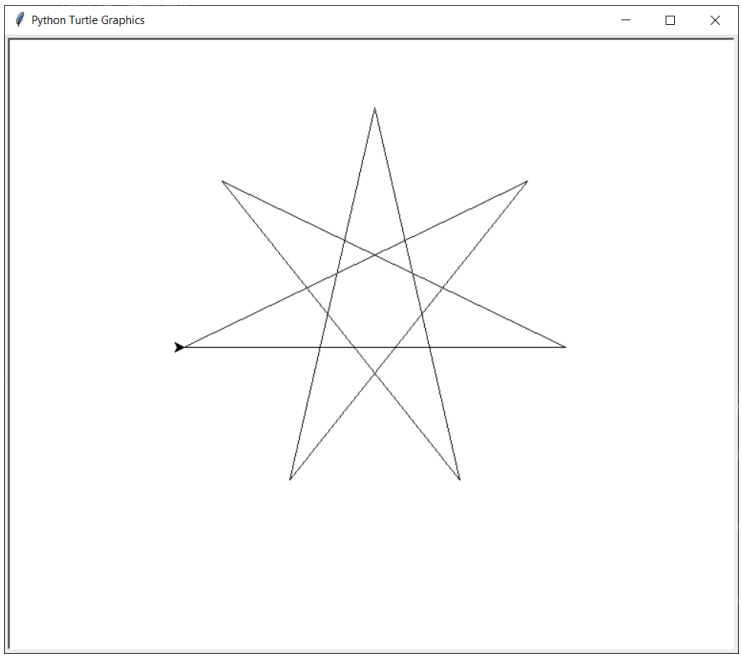


Figure 3: Sample output for $n = 14$.

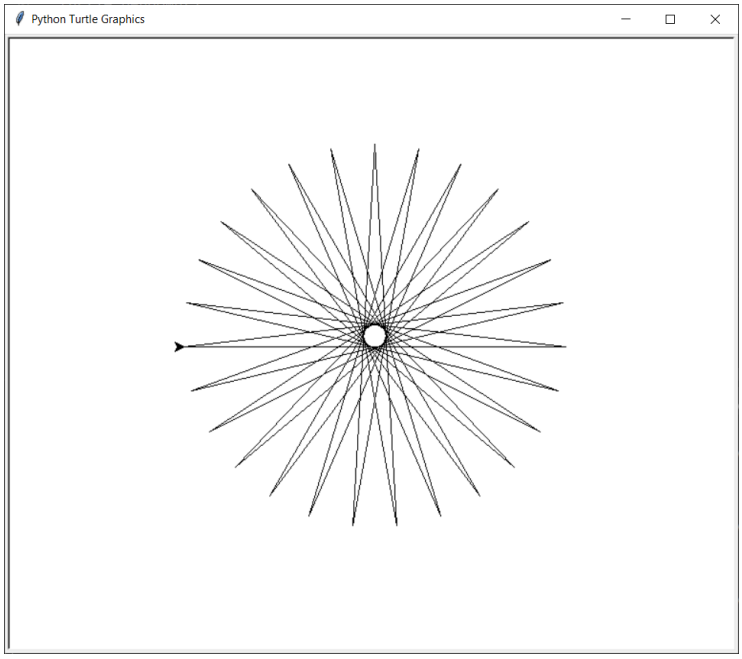


Figure 4: Sample output for $n = 27$.