

Practical 5

Programming in Security


In this practical, you will ...

- Practice using Python Files, to read and store persistent data
- Practice using Python Classes to create and manipulate objects
- Using some of Python pre-written modules
- Writing your own Python modules

Python Files Practice

1. Reading and Writing to files

To read and write properly to files, you will need to understand your project directory structures. Also note that directory structures of different operating systems would likely differ. Let us start by importing the Python "os" module to see your current working directory by using the `getcwd()` method. In the example below, the working directory is **D:\Acad\AY1920Sem2\PSEC\Practical5**. Hence when you try to access your file, it will be relative to this directory.

```
Practical5 >  Q1readwrite.py > ...  
1  import os  
2  print(os.getcwd())  
3  |  
PROBLEMS  OUTPUT  DEBUG CONSOLE  
D:\Acad\AY1920Sem2\PSEC\Practical5
```

Hence, create a files subdirectory and download the files from blackboard into the subfolder. We are ready to start accessing the files.

```
D:\Acad\AY1920Sem2\PSEC\Practical5\files>dir  
Volume in drive D is DATA  
Volume Serial Number is E037-A3D0  
  
Directory of D:\Acad\AY1920Sem2\PSEC\Practical5\files  
  
01/10/2019  04:30 PM    <DIR>          .  
01/10/2019  04:30 PM    <DIR>          ..  
01/10/2019  04:33 PM                11 notes.txt  
01/10/2019  04:30 PM                81 students.txt
```

Type in the following test codes to test the reading and writing of files. Output is on the right.

```
f=open("files\\students.txt")
for line in f:
    print(line)
f.close()

with open("files\\notes.txt", 'w') as f:
    f.write('Hello World')

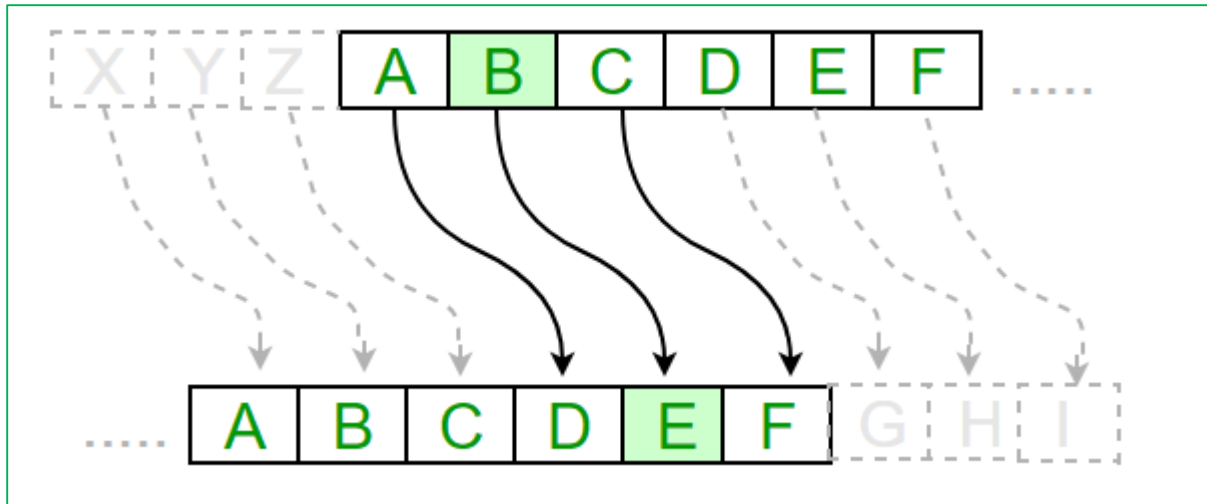
with open("files\\notes.txt") as f:
    for line in f:
        print(line)
```

```
p1234567,Junie Tan,B,B,C,F
p1234488,Kayla Lim,B,A,A,B
p1234545,Lyon Ong,A,B,C,B
Hello World
```

2. Once you are comfortable with accessing files, we can come to the fun stuff. Let's write a simple encode program to encode the contents of the 'secrets.txt' file. Define an encode (text, key=1) function that takes in a clear text (text) and a key (default value is 1). This function will implement a basic Caesar Cipher that offsets each character in the text by 'key'. (what?? Find out more: <http://practicalcryptography.com/ciphers/caesar-cipher/>)

Introduction to ASCII and Ciphers

The ASCII system provides a way to map characters to a numerical value. In this case, we are only concerned with lowercase characters from a-z (the characters in your cleaned string). We are not concerned with encoding spaces in this problem. The full ASCII table can be found here: <http://www.asciitable.com/>. The ASCII code for the character 'a' is 97 and the ASCII code for the character 'z' is 122, with the other letters falling in between those values. You can use the Python function `ord(c)` to convert a character `c` to its ASCII representation. `chr()`. The `chr(x)` function converts an ASCII integer value `x` to its corresponding ASCII symbol. For instance, `chr(ord('a'))` would return 'a', so this provides a mapping from numbers into characters.



This seemingly simple function can become quite complex so let us start with the basic encode. Open the file and pass the single line of text in the secrets.txt file into your encode function, using the default key of 1. The following code should work to call on your encode function and return the output given on the right.

```
# Code to call your encode function
f=open("files\\secrets.txt")
text=f.readline()
print(text)
cipher=encode(text)
print(cipher)
f.close()
```

Output of the code:

```
this is a secret
uijt!jt!b!tfdsfu
```

Subsequently implement the following:

- Modify encode to keep spaces and multiple lines intact (ie your cipher should only consist of lower case characters). You can assume, for simplicity, that there are no punctuation marks.
- Modify encode to handle upper case characters (convert any upper case to lower case before encoding) and handle a key of 100, while only keeping the lower case alphabets. Remember the cipher will only consist of lower case characters, spaces and new lines. (ASCII values of lower case characters ranges from 97-122)
- Write a method to put the encoded string, cipher, into a superSecret.txt file
- Write a decode function to read superSecret.txt, use the key to retrieve the original message

Python Classes Practice

3. Creating Students

- From Qn 1, you were able to read the "students.txt" file. We can now make use of this input file to generate Student objects. To start, you will need to code a Student class. Follow the slides in Topic 5 slide 12 to code a Student class in Python.
- Test the Student class by creating some instances of student objects. You can follow the codes in the slides.
- Use the data stored in "students.txt" file to create the student objects, to be stored in a List called classlist, instead of hardcoding the information like you did in b).

You will need to read the file, split the information read in by lines then commas ",". And use the data to create the student objects. Note the newline characters at the end of file! (Hint: remember you can use strip() method for strings to remove whitespaces). Hence, based on the "students.txt" file, you will get this output:

```
for s1 in classlist:  
    print(s1.name,s1.admin,s1.grades)
```

Will give:

```
Junie Tan p1234567 ['B', 'B', 'C', 'F']  
Kayla Lim p1234488 ['B', 'A', 'A', 'B']  
Lyon Ong p1234545 ['A', 'B', 'C', 'B']
```

- Code an **add_grade** method to take in and add on a list of grades into the student's existing list of grades. For example, running:

```
grades=['C','F']  
classlist[0].add_grade(grades)  
print(f"{classlist[0].name} scored {classlist[0].grades}")
```

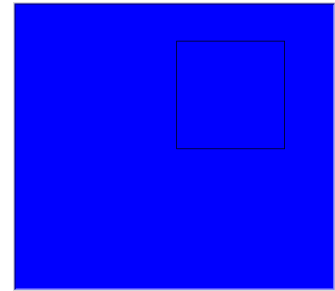
Will give:

```
Junie Tan scored ['B', 'B', 'C', 'F', 'C', 'F']
```

- Code a method to store the updated data back into the "students.txt" file.
- Code a method to compute the students' cumulative GPA, using SP's grading system. If you do not know how your GPA is computed, refer to your SP Student Handbook: <https://www.sp.edu.sg/sp/student-services/osc-overview/student-handbook/grading-system>
- Code a method to aggregate the students' grades by returning a dictionary of counts for each grade the student scored.

Python Turtle Practice

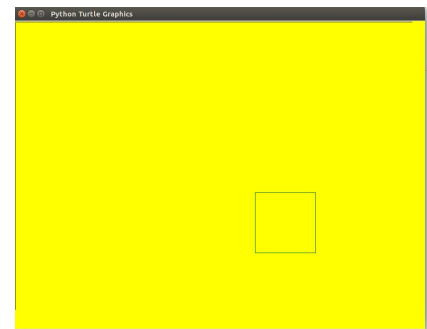
4. Try using the Python Turtle class from Topic 5 slides (slide 28). Explore and study the usages and purposes the Turtle class functions like `up()`, `down()`, `left()`, `right()`, `fd()`, `shape()`, `setx()`, `sety()`, `goto()`, `showturtle()`, `hideturtle()`, `speed()`. Also test out about Screen class functions like `bgcolor()`, `setup()`, `exitonclick()`, `screensize()`, `clear()`, `bye()`



5. Implement a `draw_square(pen, length)` function that takes a pen (Turtle) and a length (int).

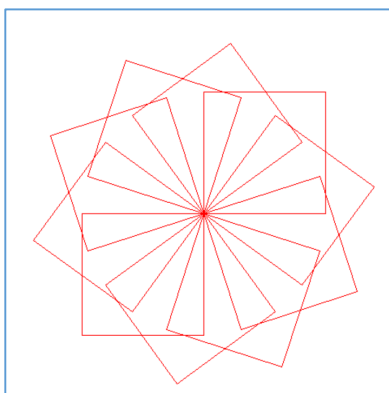
By using the Pen (an instance of a Turtle object) carry out the draw a square operation. The length of the square is specified by the argument, length.

Here is a sample run of the `draw_square (pen, length)` with its corresponding output.



6. Implement a function `draw_circle_with_sq(sc,pen,x,y,length,bgcolor,pencolor,steps)` where:
- `sc` defines the default screen instance.
 - `pen` defines the usable turtle instance.
 - `x, y` define the centre position of the circle. (default is 0,0)
 - `length` defines the length of the squares. (default is 150)
 - `bgcolor`, `pencolor` is self-explained. (default is "white", "red")
 - `steps` defines the number of squares to be used to draw the circle. (default is 10)

This function will make use of the `draw_square` function to produce a simple 2D computer art. For example, the result below is generated with the following instructions:



```
tur = Turtle()
win = Screen()
win.screensize(700,700)
win.setup(700,700)
draw_circle_with_sq(win,tur)
```

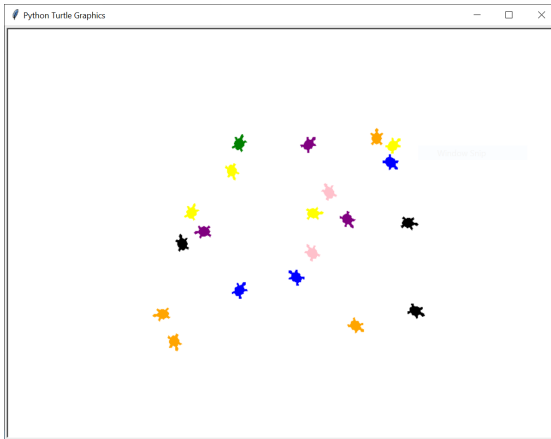
Python Modules Practice

Implement some of Python pre-written modules.

7. Code a simple number guessing game to make use of the Python random and time modules using the following guidelines. A sample output is also provided below.
 - a) Import the two modules
 - b) Set a start and end limit of the random number to be generated
 - c) Greet the user and allow the user to start the timer running
 - d) Generate a random number using the limits in b) and start the timer.
 - e) Loop to prompt user for his guesses.
 - f) When user guesses correctly, display the time spent and number of guess made.

```
Welcome to Python Guessing Game! Press Enter to start the timer!  
Random number (1-100) has been generated! Timer started!  
Input your guess: 55  
Your guess is too large! Guess something less than 55  
Input your guess: 44  
Your guess is too small! Guess something more than 44  
Input your guess: 50  
Your guess is too small! Guess something more than 50  
Input your guess: 53  
Your guess is too large! Guess something less than 53  
Input your guess: 52  
  
You got it! Answer is 52! You took 5 guesses and 19.55 seconds!
```

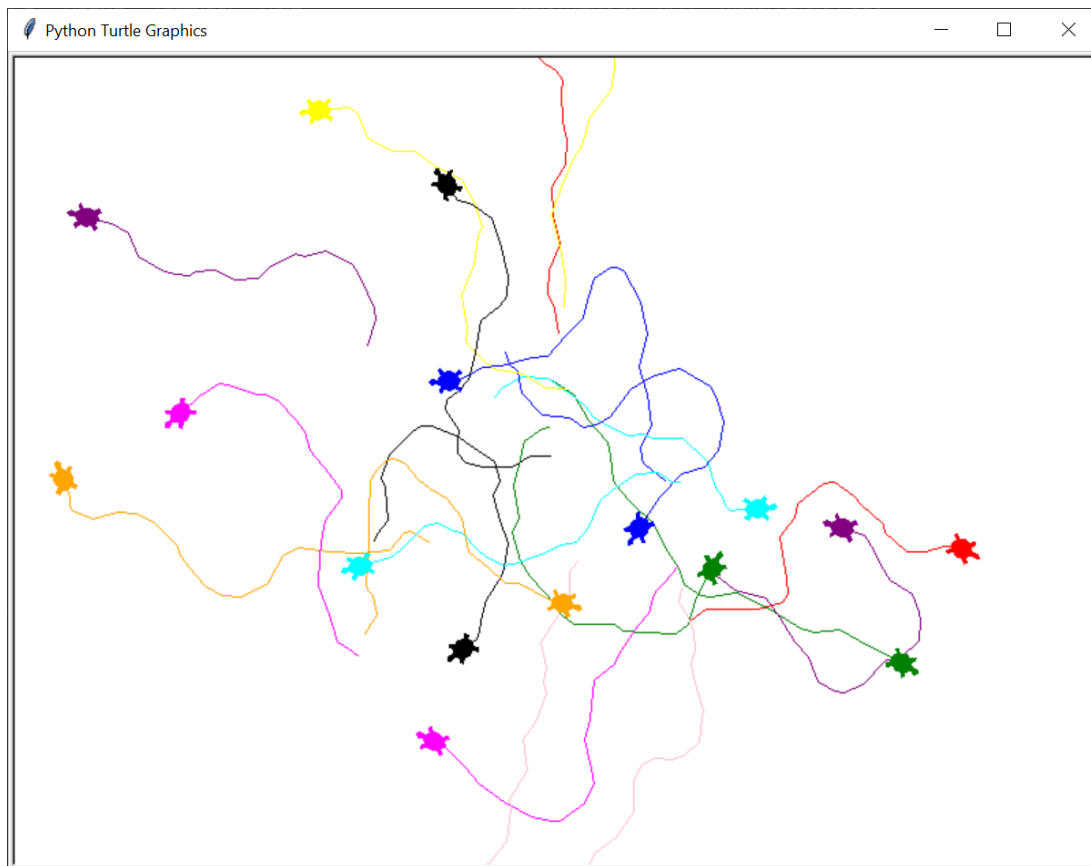
Ninja Turtles Optional Question



Create a little animated screen of turtles trotting around the screen using the given `ninjaSkeleton.py` file.

In this exercise, you will implement a `Ninja` class which inherits from the `Turtle` class. Your main program will create 20 copies of the `Ninja` instances of random colours, and let them wandering (walking in random directions and speeds) on the screen, each of them will stop and disappear from the screen when it is stepping outside of the borders of the screen.

Copy the skeleton `ninjaSkeleton.py` code to complete implementation of the `ninjaTurtle.py` class (based on the instructions found at the inline comments. Run and de-stress by watching the little animation play.



Ninja Turtle Artwork

-- End --