**ITCS-5102 Term Project Part I**

**Language: Python**

**Team 14**

**Nilanjan Mhatre - 801045013**

**Dr. Dewan Ahmed**

The paradigm of the language

* Python is a high-level programming language
* It supports
  + **Imperative:** In Python, programs can be written as a set of commands one after the other, where each command changes the state of execution
  + **Procedural:** Python supports defining procedures or functions or subroutines, that contain a series of steps to be carried out. Any procedure can be called at any point during the program’s execution. Although, these subroutines won’t be executed if they are not called.
  + **Functional:** Functional programming is supported by Python, which is similar to procedural, but does include the feature of changing the state of execution or having a series of statements, but only rely on evaluation of expressions
  + **Object-Oriented:** Python allows creation of classes and encapsulation of methods and data inside a class. Other OOPS features include Inheritance, Polymorphism and abstraction

History of “PYTHON”

* Python was created by Guido Van Rossum in 1989 at CWI in Netherlands, when he published the code as version 0.9.0
* It was built to be an object-oriented programming language from the beginning
* But, it also supported procedural programming, i.e. unlike Java, it is not purely an object-oriented language
* Later, with contributions and patches from others like a Lisp hacker, Python reached its version 1.0 that included new features: lambda, map, filter, reduce
* There were many releases in the due course including the ‘Macintosh only’ 0.9.5 version
* Python 1.4 also acquired new features like the Modula-3 keyword arguments and support for complex numbers
* Python gained popularity because of its clean coding capability and ease of learning the language with its non-complex syntax
* In 2000, the Python core development team moved to BeOpen.com to form Python labs that resulted in the release of Python 2.0
* It had features of list comprehension, cycle-detecting garbage collector and Unicode
* Python 2.2 had a major change that combined the types written in C and classes in Python into one hierarchy
* Python 3.0 rectified certain issues in the language and made a few syntax changes
* Some of them are changing the print statement to print function, moving only ‘reduce’ out of the namespace into functools, using only Unicode for text strings, adding support for optional function annotations and also changing integer divisions (e.g. In 2.x, 5/2=2, while in 3.x 5/2=2.5 and 5//2=2)
* This broke the backwards compatibility, which is why tools must be used to convert programs written in Python2.x into Python 3.x
* All changes are now being operated on current 3.x series
* Python 3.6 had changes in the new UTF-mode and Python 3.7.0b1 (PEP 540) adds a new "UTF-8 Mode" (and overrides POSIX locale).

**Antecedents**

* 1. ABC
  2. ALGOL
  3. C
  4. C++
  5. CLU
  6. Dylan
  7. Haskell
  8. Icon
  9. Java
  10. Lisp
  11. Modula-3
  12. Perl

Reserved words, primitive data types, structured types

**Reserved words:**

* + - False
    - None
    - True
    - and
    - as
    - assert
    - break
    - class
    - continue
    - def
    - del
    - elif
    - else
    - except
    - finally
    - for
    - from
    - global
    - if
    - import
    - in
    - is
    - lambda
    - nonlocal
    - not
    - or
    - pass
    - raise
    - return
    - try
    - while
    - with
    - yield

**Primitive data types:**

**Structured types:**

|  |  |
| --- | --- |
| * list * tuple | * dictionary * set |

* Boolean
* Numeric: int, long, float, complex

Syntax

A python program is a combination of lines, where a newline serves as a token for termination of a single statement. Braces or brackets are not required to define a block of code. Lines that contain only whitespace along with an optional comment are considered blank lines and are ignored by the interpreter.

Blocks are defined with whitespace (spaces and tabs) rather than the common counterpart, braces ({}). Usability is facilitated by allowing arbitrary yet consistent spacing.

Console output is done using “print” function, which was a keyword in previous versions.

 Output: 

The Python interpreter parses a line as a series of characters, using , until a ‘newline` is found.

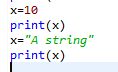
Although, if a single statement has to be written in multiple lines, a backslash “\” character is used. On the contrary, multiple statements can be combined in a single statement separated by a semi-colon “;”.

 Output: 

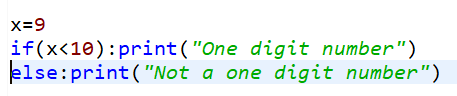
Variables in Python do not require declaration. They are used for assignment directly.

*x=10*

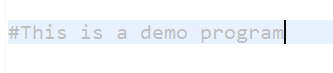
Also, a variable can hold any type of value, which can vary in runtime. It is not restricted to a single data type.

 Output: 

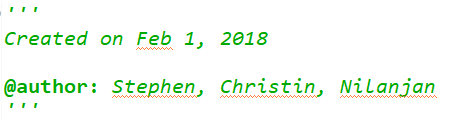
Conditional statements are written in different lines with one block for each condition to be true. One block will terminate until the indentation is changed.

 Output: 

Comments are mentioned using the hash “#” symbol, which can appear at any point in the program, except in a string. The interpreter ignores the characters after “#” till the end of the line.

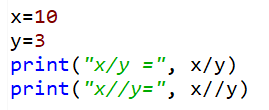


A block of comment consisting of multiple lines can be expressed having it surrounded by 3 single quotes on each side.



A function is written using the keyword “def”, which will not be executed until it is called. Also, Python follows a top-down approach, which is why a function has to be defined before it is called, otherwise, the interpreter will not recognize that function.

Operations in Python follow “infix” notation, and brackets are used for specifying priority. Python provides two operators for “division”. The “/” operator gives a decimal result, while the “//” operator is a special operator that gives only the integer result.

 Output: 

The basic control abstractions of the language (loops, conditional controls, etc.)

1. if/elif/else statements
2. while/for loops (in range, in elements of structure)
3. iterators (
4. list comprehension expressions *(enhanced for loop?)*
5. break/continue/pass statements
6. try/except/finally statements

How the language handles abstraction (including functions, procedures, objects, modules, etc.)

With Python’s focus on readability, many complex components (whether a function, object, or package) are drastically simplified, sometimes into a single line. Since user’s can define their own components, this abstraction can be broadened to further complex ideas.

* Data Abstraction: -
  + Python allows storing data in variables, and supports numerous data types, including the primitive data types and non-primitive data types like “list” and “tuple”
  + A “list” can hold several objects of the same data type, while a tuple can hold objects of different data types

e.g. list1=[1,2,3,4]

student=(“Miranda”, “Jones”, 3, “CS”, “02-02-1992”)

* + It does not impose any restrictions on variables and allows not to declare the type of the variable, along with allowing the same variable to hold different data types in a single execution as shown in the syntax example
  + Python allows creation of packages that can be imported to reuse the functionality
* Control Abstraction: -
  + Python supports procedural programming, where one can write functions and reuse those functions multiple times to avoid rewriting the code
  + Using loops and conditional statements, allows the programmer to write a logic in a much easier way than the same in Assembly language

Readability, Writability and Reliability

**Writability**

* 1. **Simplicity and Orthogonality:**
     1. Python’s design emphasizes explicitness, which facilitates familiarity and utilization. A common standard in the language is to include a function within any type of unit abstraction that prints the documentation (or guide) to explain its functionality, usually including a comprehensive list of functions and attributes.
  2. **Support for Abstraction:**
     1. With the ease and speed of defining and calling a function, abstraction in this manner is supported heavily. There are also many predefined functions, some of which encapsulate complex methods.
     2. Python practices dynamic typing, which means upon assignment, a variable will determine its data type once the actual value is returned. This allows for generic typing to be done automatically, shortening code and quickening the logical process. Complex data structures also are designed as clearly as possible, and, if mutable, are easily accessed and manipulated.
  3. **Expressivity:**
     1. This is one of the main foundations of Python ideology. Being able to say more with less is hard to avoid in this language. This can be found in functions, data structures, modules, and packages. With such a high level of abstraction, intuitive expressions are increasingly efficient.

**Readability**

* 1. **Simplicity:**
     1. While it can be difficult to differentiate its structured types, there is a relatively small amount of effort required to understand the most basics concepts of the language and how they are implemented. With such a high level of abstraction, simplicity naturally follows.
     2. Feature multiplicity is supported in Python, yet the language strives to provide a single, most-obvious way to accomplish a task. Still, users are given an open range of types and functions to use based on their preferences. Examples include building structured types, analyzing data, and presenting output.
     3. Python allows users to overload practically any operator by defining predetermined functions in an object’s class. If the user desires to add two circle objects together to produce a new circle, a function can be defined with the object’s class that executes this feature. It can be decided whether to add the perimeter lengths, area, etc.
  2. **Orthogonality:**
     1. This aspect is capitalized in Python, particularly with ­dynamic typing. There is complete overlap of constructs by allowing the variable to accept any data type. Functions, especially regarding data structures, also provide some overlap, yet there is some restriction when working with multiple data types within a single function. The level of orthogonality was set to provide both straightforwardness and logic.
  3. **Data Types:**
     1. Each data type within Python has its own set of applications and functions, and while there is some overlap in these areas between types, it is relatively simple to differentiate between each. Implicit typing can add a variable degree of difficulty when reading the language.
  4. **Syntax Design:**
     1. Reserved words are modified in some form to stand out (in most IDEs), and many predefined, reused words are surrounded by underscores. This eases readability and comprehension. Removing curly brackets and semicolons further increases this.
     2. The form and meaning of the language is designed to be highly intuitive and clear. Many words explicitly represent their applications. For example, for loops in Python are significantly more understood during an unexperienced person’s first encounter with them than in other languages.

**Reliability**

1. **Type Checking:**
   * 1. Data types are checked during runtime in Python (ie. dynamically). This potentially causes difficulty for some writers since it increases the likelihood of runtime errors and, in some cases, the effort to fix type errors. A major tradeoff is the utilization of powerful language features, such as dynamic dispatch and reflection. With complex functions, dynamic typing can actually increase reliability through flexibility.
2. **Exception Handling:**
   * 1. Python handles exceptions similarly to several other languages by throwing the exception to the console, including its type, location, and description. It does this rather elegantly by providing such a clear, concise report of the error. Again, like other languages, it also allows for the writer to implement try-except clauses.
3. **Aliasing:**
   * 1. The primary example of this is Python is in the use of for loops. Data and the elements within it can be renamed for reference from within the loop body. This allows for greater simplicity and intuitiveness, which, for some users, increases reliability.
4. **Readability and Writability:**
   * 1. As mentioned before, some users will experience strengthened reliability through efficient and simplified readability and writability. This is an essential factor that lead to its popularity. Errors and logic and more easily found and followed, respectively.

The major strengths and weaknesses

The language's core philosophy is summarized by aphorisms, such as:

*“Beautiful is better than ugly*

*Explicit is better than implicit*

*Simple is better than complex*

*Complex is better than complicated*

*Readability counts”*

- Zen Python??

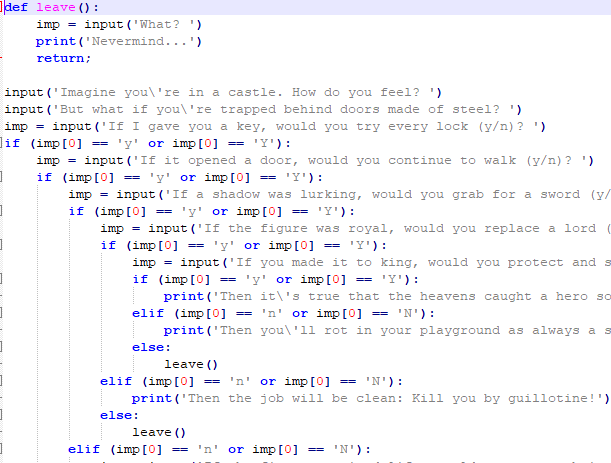
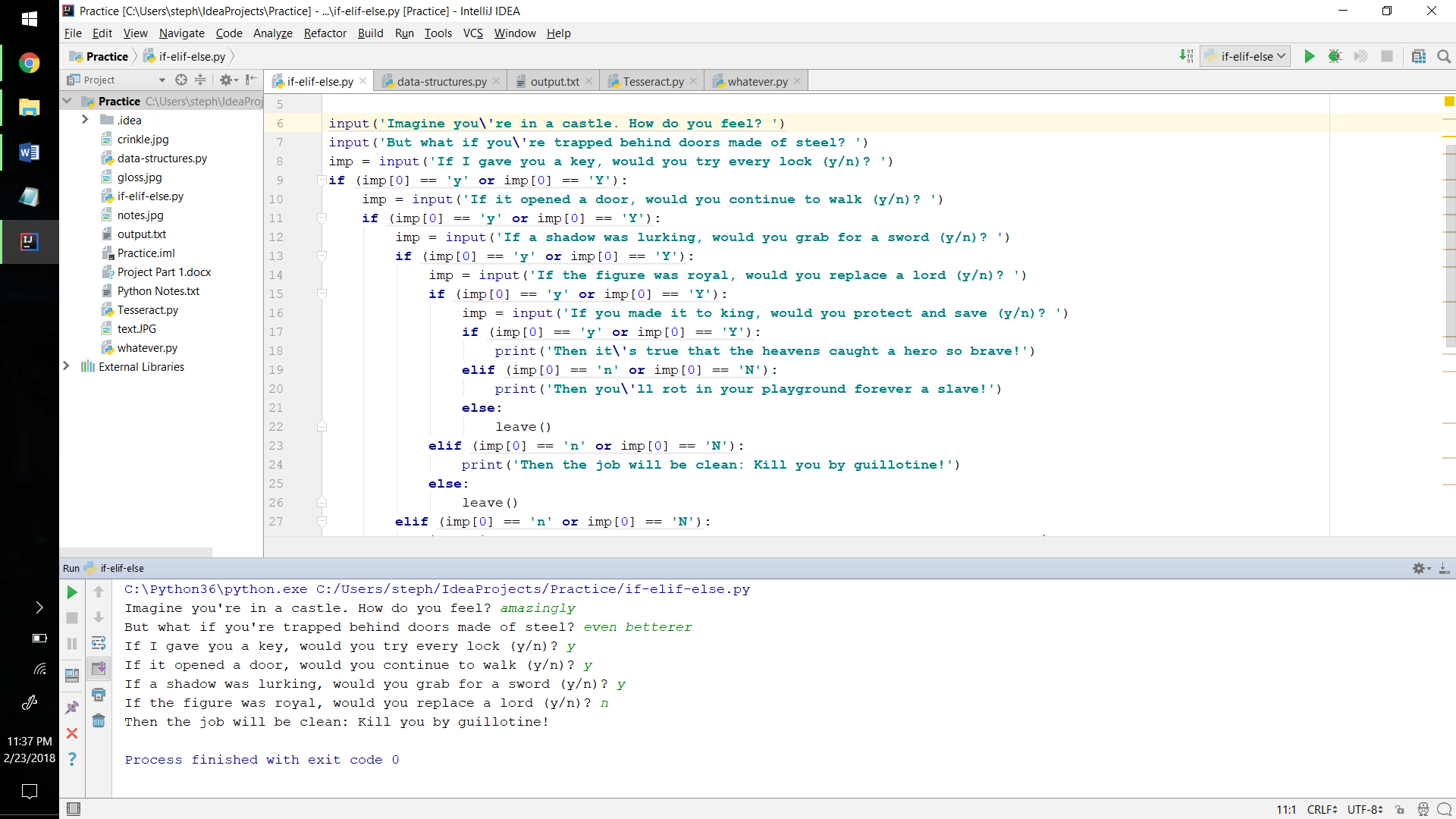
* Advantages: -
  + Python has a large standard library
  + It was designed with high extensibility
  + It is a good option for building desktop applications and servers
  + Furthermore, the syntax in Python helps the programmers to do coding in fewer steps as compared to Java or C++
  + It is highly Interactive
  + Debugging is easy in Python
  + Python integrates the Enterprise Application Integration making it easy to develop Web services using COM or CORBA with good support for XML and JSON
  + It is modular and dynamic, which is why it is a good option for scalable application
  + It supports almost every programming language paradigm
  + With these reasons, Python is determined to be a general-purpose language
* Drawbacks: -
  + A major disadvantage of the language is averagely longer testing periods, with most errors occurring at run-time, which is why it creates restrictions in design
  + In certain situations, Python has shown slower performance as it does not save time by compiling the program beforehand, and uses an interpreter to parse and run the program
  + The syntax of Python can be confusing to use, with the absence of braces and semi-colons to define
  + Python is not viable to be used in mobile computing
  + Python’s Database access layers are under developed as compared to powerful JDBC and ODBC

An overview of the programs that you included and a discussion of what language features they highlight and how the language made the programs easy/hard to implement.

**If-Elif-Else Castle Journey (*if-elif-else.py*)**

This program highlights writability and readability. With the story prewritten, organizing and writing the nested conditional statements took about 10 minutes. A user-defined function is also implemented to handle incorrect input, which further enhances both abilities. Multiple people with no coding experience understood the logic flow and syntax of the program within a single minute.

A major complication of writing complex code is incorrectly placed, missing, or extra braces. Python mitigates that immediately by only requiring indentation, and similar benefits can occur regarding no requirement of semicolons at the end of logic lines.

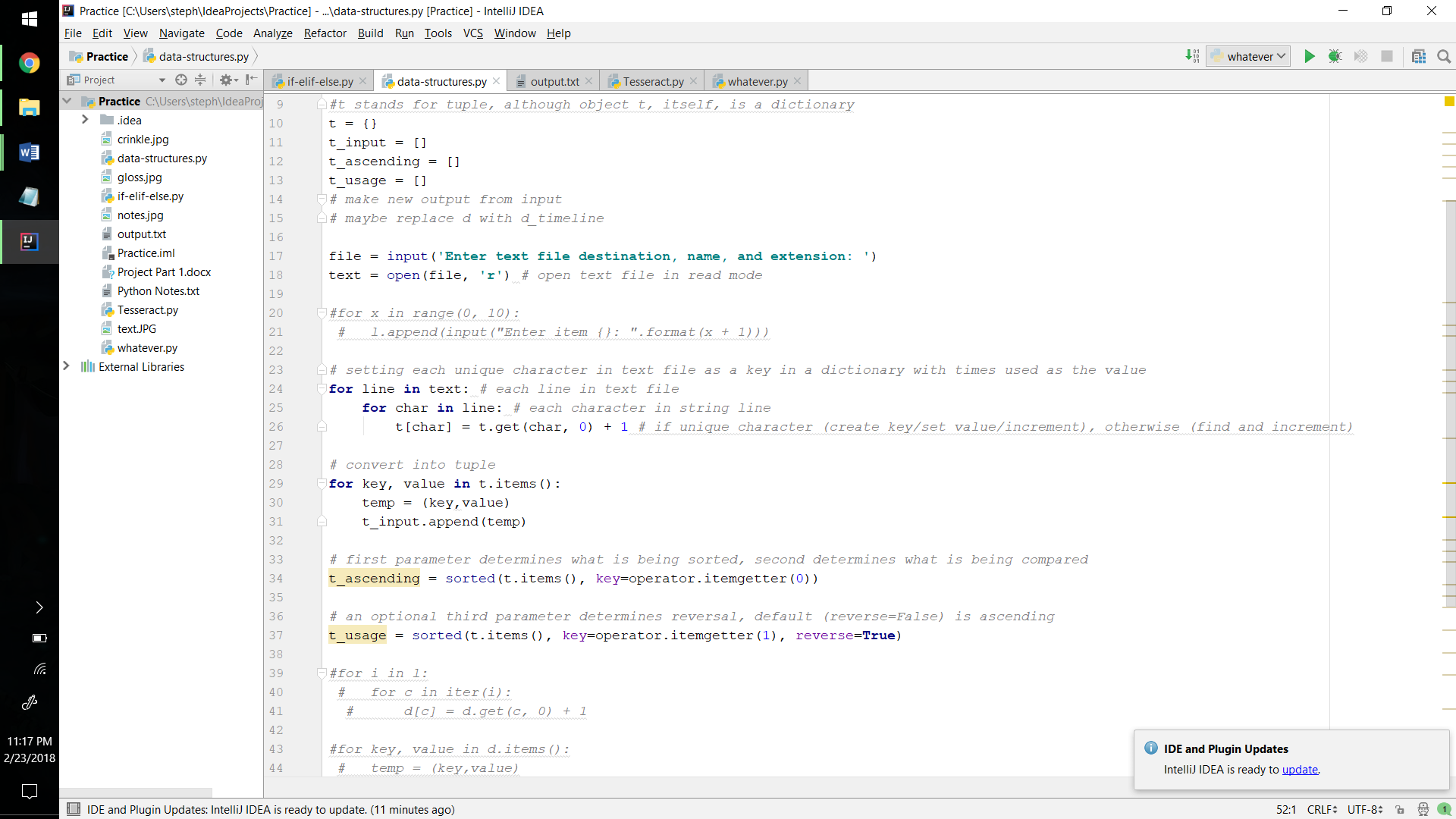


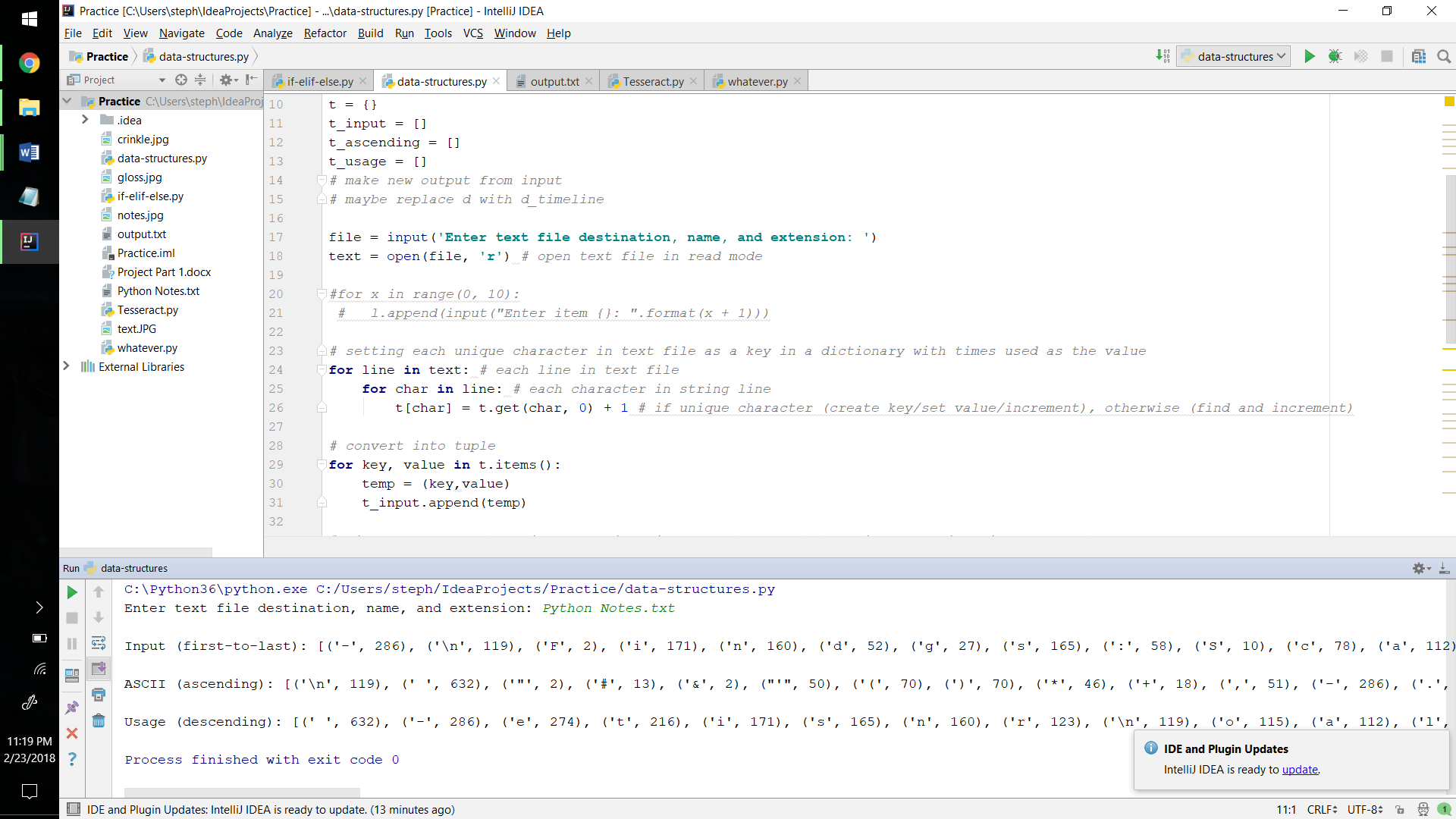
**Text Statistics (data-structures.py)**

This program highlights the ease with forming, decomposing, and manipulating data structures, all done with less code compared to Java.

A dictionary was used to store individual characters in a text file and their total frequency. The dictionary was then converted into a tuple then sorted in two different ways, resulting in a total of three tuples. Each were printed by simply passing the variable through a print statement, and the output was automatically organized and easily readable without further manipulation.

Python efficiently and conveniently provides mechanisms to work with data structures in any way, increasing writability, readability, and reliability.





Output:

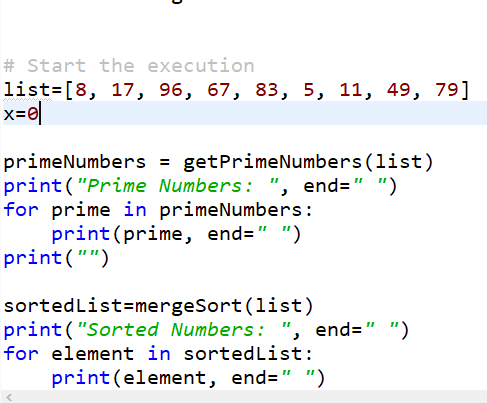
**List manipulation (ListDemo.py)**

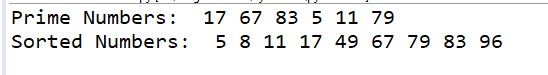
This program highlights the usage of lists and the operations that can be performed on a list.

A list has been initialized. It is then passed to the “*getPrimeNumbers”* method that will compare each element, whether it is a prime number, and will return another list which will include only prime numbers. To generate another list, “*append”* was used to add elements. The “*getPrimeNumbers”* method internally calls the “*isPrime”* method.

Merge Sorting was performed on the list, that uses a recursive method to sort the list. “*mergeSort”* is recursively called, after which “*merge”* is called to get the final sorted list.

Snap of the program





Output

References

<https://medium.com/@mindfiresolutions.usa/advantages-and-disadvantages-of-python-programming-language-fd0b394f2121>

<https://en.wikipedia.org/wiki/Python_(programming_language)>

<https://www.protechtraining.com/content/python_fundamentals_tutorial-functions>

<https://docs.python.org/3/>

<https://www.safaribooksonline.com/library/view/python-in-a/0596001886/ch04s09.html>

<https://python.swaroopch.com/control_flow.html>

Advanced Program

“Bows and Balloons”, a game made in Pygame library supporting Python

* Pygame allows construction of Sprites which is an object taking up space in the window
* Sprites are managed in the memory efficiently to improve performance
* Sprites can be programmed to move from one location to the other or change behavior
* Collision of two Sprites can be detected
* Pygame can take input events like, mouse pointer, mouse clicks and key presses
* Using simple Python code, the state of the game can be controlled

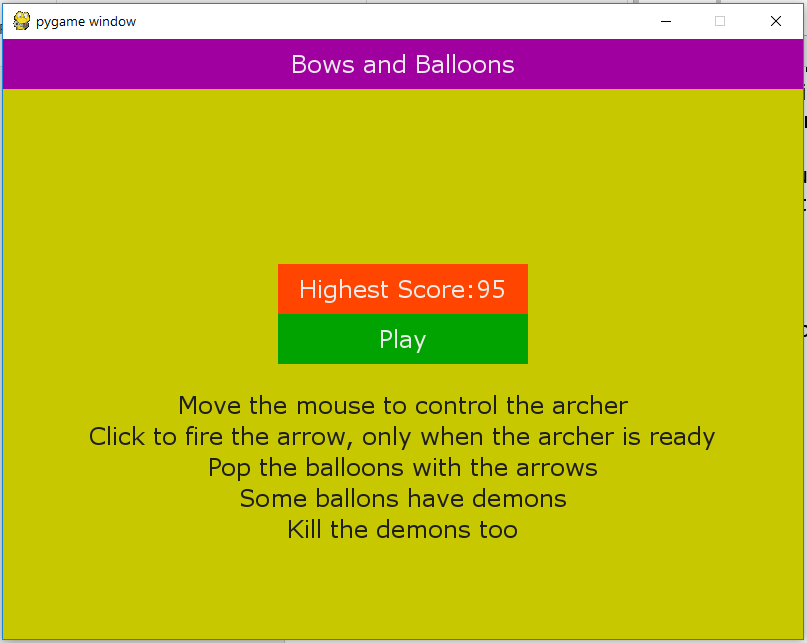
Run the code: -

Import in eclipse and run “bows\_balloons.py”

OR

Open command prompt or terminal on the directory and fire the command,

***>>py bows\_balloons.py***



Highest Score is stored in the file “highest\_score.txt”

The latest score is compared, to get update the highest score in the file, by input/output operations on that file.

The **HomePage** class shows how the homepage is designed with game instructions.

* + - * Click on “Play”

An **Archer** sprite is created that is controlled by the mouse pointer.

It has multiple states. The **Archer** can fire the arrow only when it is ready.

**Balloon** is another sprite. Each Sprite has a dedicated class created.

**Balloons** will be spawned using an algorithm.

The **Config** class stores all the game configuration, which are used in the spawning logic.

The **Archer** must fire the arrow at the precise time that it should hit the **Balloons**.

* + - * Click anywhere in the window to fire the arrow

Each **Balloon** popped will earn +10 points.

While spawning a **Balloon**, the algorithm randomly can put a **Demon** inside the **Balloon**, which if popped, will free the demon, to charge towards the **Archer**.

If the **Demon** hits the **Archer**, -15 points are deducted. No points if **Demon** misses the **Archer.**

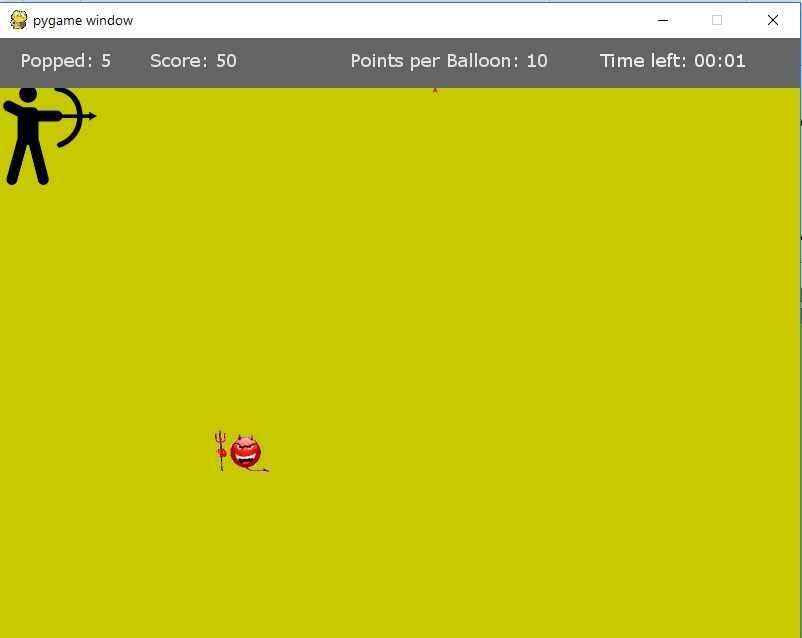
Alternatively, the **Archer** can kill the **Demon** by arrow, to earn +5 points.

This continues till the timer ends. So the objective is to get maximum score.

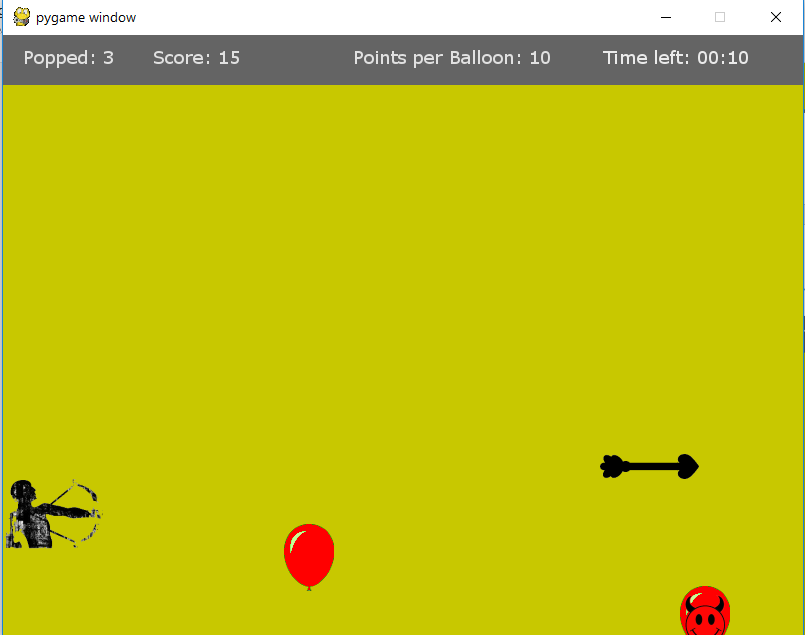
The **Balloon** or the **Demon** is popped only when the **head of the arrow**, not the tail hits it, meaning that, head is a different Sprite, which is checked for collision with other Sprites.



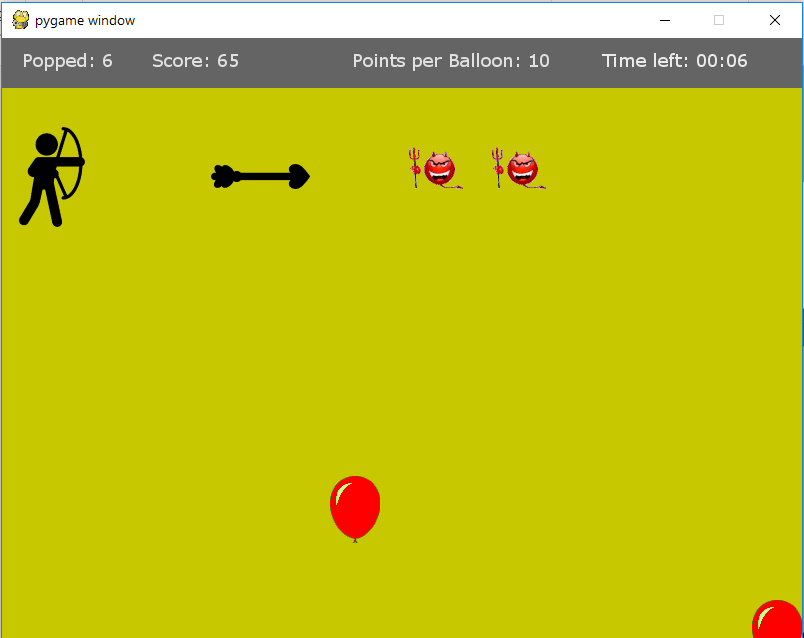
Demon Balloon popped



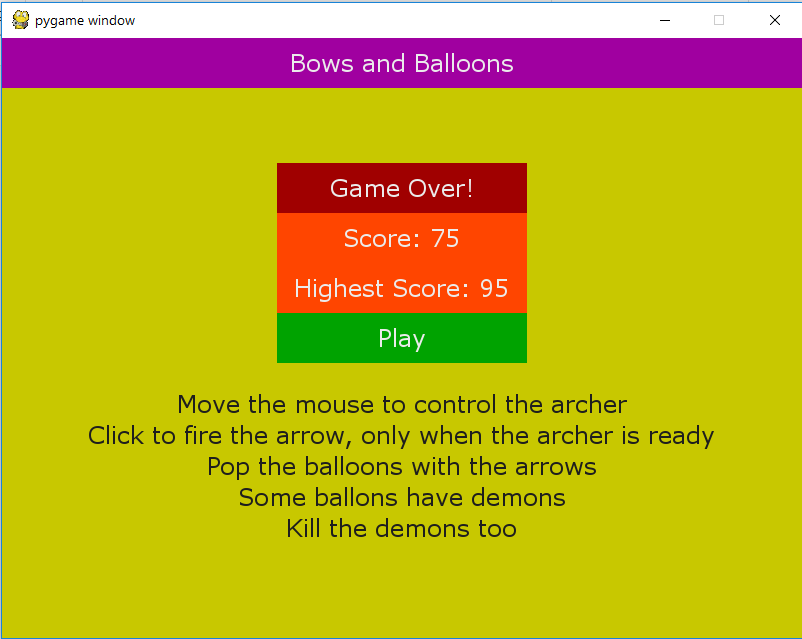
Demon hits Archer



Arrow hits demons



Current and Highest Score



Code: -

To start Pygame, use pygame.init()

Then, start an endless loop, that will terminate only on some specific action, e.g. exit button clicked

Check events and status of the game, to display only those Sprites that are relevant.

Set the screen width. Set the Sprites size, with image if necessary or with text and specify their starting position, which is a co-ordinate system of (x, y) in pixels of the screen.

Handle the events and control the Sprites, by moving them or changing their state.

References

<https://nerdparadise.com/programming/pygame/part1>

<https://www.pygame.org/wiki/tutorials>

<http://programarcadegames.com/>