Day - 24

**File I/O and Directory**

Local file I/O and Directories, High score record in Snake game,

Automate Letter/Application

**24.1 Snake Game - High Score (file I/O).**

Update the Snake -Game

Scoreboard:

1. Add ***high\_score*** in ***\_\_init\_\_()***
2. Replace ***game\_over()*** with ***reset\_score(),*** pass the score value to ***high\_score***.

* Order matters of resetting score

1. Modify the ***update\_score()*** and ***increase\_score()***
2. Call ***upade\_score()*** in ***reset\_score()***

Main:

1. Replace ***game\_over()***
2. Add snake ***reset\_sanke()***

Snake:

1. Define ***snake\_reset()***: Clear segments, create snake, head. Remove dead snake off the screen

* **clear()** for EMPTY the ARRAY: Instead of clearing the ***turtles*** (which is in ***Turtle*** class), we just empty the array.

To empty an array use ***clear()*** method.

|  |  |  |
| --- | --- | --- |
| scoreboard.py | snake\_main.py | snake.py |
| **import** turtle  ALIGNMENT = 'center'  FONT = ('Arial', 16, 'normal')  **class** Scoreboard(turtle**.**Turtle):  **def** **\_\_init\_\_**(self):  **super**()**.\_\_init\_\_**()  . . .  **self.**score = 0  **self.**high\_score = 0  **self.update\_scoreboard**()      #*def gameover(self):*      #*self.goto(0 , 0)*      #*self.write(f"GAME OVER", move= False, align= ALIGNMENT, font= FONT)*  **def** **reset\_score**(self):          #*Updating high\_score*  **if** **self.**score **>** **self.**high\_score:  **self.**high\_score = **self.**score  **self.**score = 0  **self.update\_scoreboard**()  **def** **update\_scoreboard**(self):  **self.clear**()  **self.write**(f"Score : {**self.**score} \t High Score : {**self.**high\_score}", move= **False**, align= ALIGNMENT, font= FONT)  **def** **icrease\_score**(self):  **self.**score += 1  **self.update\_scoreboard**()    #*python scoreboard.py* | **while** game\_on:  . . . . .      #*Wall Collision: Stop the snake at boundary.*  **if** (snAKe**.**head**.xcor**() **>=** 360) **or** (snAKe**.**head**.xcor**() **<=** -360) **or** (snAKe**.**head**.ycor**() **>=** 310) **or** (snAKe**.**head**.ycor**() **<=** -310):          scrBrd**.reset\_score**()          snAKe**.reset\_snake**()          #*game\_on = False*    . . . .  **for** sQr\_segmnt **in** snAKe**.**snake[1 : ]:  **if** snAKe**.**head**.distance**(sQr\_segmnt) **<** 10 :              scrBrd**.reset\_score**()              snAKe**.reset\_snake**()              #*game\_on = False*    #*Screen doesn't disappear autometically*  scr**.exitonclick**()  #*python snake\_main.py* | **. . . . .**  **def** **reset\_snake**(self):          #*moving dead snake*  **for** seg **in** **self.**snake:              seg**.goto**(1000, 1000)    **#*Clearing the array***  **self.**snake**.clear**()  **self.create\_sanke**()  **self.**head = **self.**snake[0]          #*self.move() not needed while-loop in main will do the job*  **def** **up**(self):  **if** **self.**head**.heading**() **!=** DOWN:  **self.**head**.setheading**(UP)  **. . . . .**  #*python snake.py* |

**24.2 File I/O using the** "with" **Keyword**

**def** open(file: Union[str, bytes, int], mode: str=..., buffering: int=..., encoding: Optional[str]=..., errors: Optional[str]=..., newline: Optional[str]=..., closefd: bool=..., opener: Optional[Callable[[str, int], int]]=...)

* Open a file: Open file and return a stream. Raise ***OSError*** upon failure.
* ***file*** is either a ***text*** or ***byte string*** giving the name (and the ***path*** if the file isn't in the current working directory) of the file to be opened.
* ***mode*** is an optional string that specifies the mode in which the file is opened.
* It ***defaults*** to ***'r'*** which means ***open*** for ***reading*** in text mode.
* Other common values are ***'w'*** for ***writing*** (truncating the file if it already exists),
* ***'x'*** for creating and writing to a ***new*** file, and
* ***'a'*** for appending (which on some Unix systems, means that all writes append to the end of the file regardless of the current seek position).
* Following are the various File Modes in Python: Recall 18.6 File I/O in Python

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Mode** | **Description** | | **‘r’** | This is the default mode. It Opens file for reading. | | **‘w’** | This Mode Opens file for writing. If file does not exist, it creates a new file. If file exists it truncates the file. | | **‘x’** | Creates a new file. If file already exists, the operation fails. | | **‘a’** | Open file in append mode. If file does not exist, it creates a new file. | | **‘t’** | This is the default mode. It opens in text mode. | | **‘b’** | This opens in binary mode. | | **‘+’** | This will open a file for reading and writing (updating)  **'a+'** or **'w+'** | | Step 1) **Open the .txt file**  f= **open**("guru99.txt","w+")  Step 2) **Enter data into the file**  for i in range(10):  f.**write**("This is line %d\r\n" % (i+1))  or, f.**read**()  Step 3) **Close the file instance**  f.**close**() |

#*python file\_io\_demo.py*

opend\_file = **open**(file = "my\_file\_test.txt")

content = opend\_file**.read**()

**print**(content)

#*Close the file to free up the memory*

opend\_file**.close**()

* Closing a file: Though Python automatically closes a file if the reference *object* of the *file* is *allocated* to *another* *file*, it is a standard practice to close an opened file (using ***close()*** ) as a ***closed file reduces the risk of being unwarrantedly modified or read***.
* Auto closing: Use following way to open a file and after exiting this block the file will automatically closed.

#*Auto closing*

**with** **open**(file = "my\_file\_test.txt") **as** opend\_file:

    content = opend\_file**.read**()

**print**(content)

#*python file\_io\_demo.py*

"***with***" keyword and ***as*** for alia/var\_name.

* Writing to a file: Use the ***write()*** method.

#*Auto closing. And use the "mode".*

**with** **open**(file = "my\_file\_test.txt", mode= "a") **as** opend\_file:

    opend\_file**.write**(f"\nHaee o Silva Awayyy ")

**24.3 Challenge Read and Write the High Score to a File in Snake**

Write file inside **reset\_score**() and read file inside **update\_scoreboard**()

**def** **reset\_score**(self):

        #*Updating high\_score*

**if** **self.**score **>** **self.**high\_score:

**self.**high\_score = **self.**score

            #*record = open(file = "high\_score\_record.txt", mode="a")*

            record = **open**(file = "high\_score\_record.txt", mode="w")

            record**.write**(**str**(**self.**high\_score))

            record**.close**()

**self.**score = 0

**self.update\_scoreboard**()

**def** **update\_scoreboard**(self):

**self.clear**()

        retrive\_record = **open**(file= "high\_score\_record.txt", mode="r")

        data = retrive\_record**.read**()

**print**(data)

        retrive\_record**.close**()

        #*self.high\_score = int(data[-1])*

**self.**high\_score = **int**(data)

**self.write**(f"Score : {**self.**score} \t High Score : {**self.**high\_score}", move= **False**, align= ALIGNMENT, font= FONT)

* However instead of read file inside **update\_scoreboard**(), we can do this inside **\_\_init\_\_()** as below:

**def** **\_\_init\_\_**(self):

**super**()**.\_\_init\_\_**()

**self.hideturtle**()

**self.color**("#3c412c")

**self.penup**()

**self.speed**("fastest")

**self.goto**(0, 310)

**self.**score = 0

**with** **open**(file= "high\_score\_record.txt", mode="r") **as** data:

**self.**high\_score = **int**(data**.read**())

**self.update\_scoreboard**()

**24.4 Understand Relative and Absolute File Paths**

* Absolute File Paths: Absolute File Paths is always relative to the root.

L:\1\_Development\z-codes-2.1-PYTHON\MLp1\_py\_ang\_100dy\_day24\_File\_OI\_dir\\_\_pycache\_\_

* Relative File Paths: If we are currently inside L:\1\_Development\z-codes-2.1-PYTHON\MLp1\_py\_ang\_100dy\_day24\_File\_OI\_dir as working directory, then in our ***.py*** file we can specify the relative file path as:
* ***./\_\_pycache\_\_/file\_name*** here "**./**" simplify look in the current folder.
* To go to the parent folder use "**../**"
* In our case: In previous codes we've actually used relative-paths as, **open**(file = "**high\_score\_record.txt**", mode="w")
* Which is short form of **open**(file = "**./high\_score\_record.txt**", mode="w") both have same meaning
* Exercise 24.1: Access the files in our snake game at different directory. Need to go to *parent* *folder* and access *data/file*

Relative:

#*----------- Relative path -------------*

**with** **open**(file= "**../data/high\_score\_record.txt**", mode="r") **as** data:

**self.**high\_score = **int**(data**.read**())

record = **open**(file = "../data/high\_score\_record.txt", mode="w")

Absolute:

#*----------- Absolute path -------------*

**with** **open**(file= "L:/1\_Development/z-codes-2.1-PYTHON/MLp1\_py\_ang\_100dy\_day24\_File\_OI\_dir/data/high\_score\_record.txt", mode="r") **as** data:

* Jumping MULTIPLE LEVEL in relative file path: Just use "../" multiple times. For example:

"../../data/file\_name"

**24.5 Mail Merge Challenge**

Practiced Version

#*TODO: Create a letter using starting\_letter.docx*

#*for each name in invited\_names.txt*

#*Replace the [name] placeholder with the actual name.*

#*Save the letters in the folder "ReadyToSend".*

#*Create name-list from the  invited\_names.txt*

names\_file = **open**(file="./Input/Names/invited\_names.txt", mode="r")

name\_list = names\_file**.readlines**()

names\_file**.close**()

#*Read starting\_letter.docx*

letter\_line\_file = **open**(file="./Input/Letters/starting\_letter.docx", mode="r")

line\_list = letter\_line\_file**.readlines**()

letter\_line\_file**.close**()

#*stripping the names*

names = []

**for** i **in** **range**(0, len(name\_list)):

**if** name\_list[i] **==** "\n":

**pass**

**else**:

        nme = name\_list[i]**.strip**("\n")

        names**.append**(nme)

#*writting the letters*

**for** j **in** **range**(0, len(names)):

    line\_1 = line\_list[0]

    first\_line = line\_1**.replace**("[name]", names[j])

    write\_string = ""

    write\_string += first\_line

**for** k **in** **range**(1, len(line\_list)):

        write\_string += line\_list[k]

    #*print("\n", write\_string)*

    write\_file = **open**(file = f"./Output/ReadyToSend/{names[j]}.docx", mode="w")

    write\_file**.write**(write\_string)

    write\_file**.close**()

#*Hint1: This method will help you: https://www.w3schools.com/python/ref\_file\_readlines.asp*

    #*Hint2: This method will also help you: https://www.w3schools.com/python/ref\_string\_replace.asp*

        #*Hint3: THis method will help you: https://www.w3schools.com/python/ref\_string\_strip.asp*

#*python main.py*

**Instructors Solution**

PLACEHOLDER = "[name]"

**with** **open**("./Input/Names/invited\_names.txt") **as** names\_file:

    names = names\_file**.readlines**()

**with** **open**("./Input/Letters/starting\_letter.docx") **as** letter\_file:

    letter\_contents = letter\_file**.read**()

**for** name **in** names:

        stripped\_name = name**.strip**()

        new\_letter = letter\_contents**.replace**(PLACEHOLDER, stripped\_name)

**with** **open**(f"./Output/ReadyToSend/letter\_for\_{stripped\_name}.docx", mode="w") **as** completed\_letter:

            completed\_letter**.write**(new\_letter)

**24.6 Python File** ***readlines()* Method**

The ***readlines()*** method returns a list containing each line in the file as a list item. Syntax:

file.**readlines**(**hint**)

***hint*** (Optional): If the number of bytes returned exceed the hint number, no more lines will be returned. Default value is -1, which means all lines will be returned.

**24.7 Python String *replace()* Method**

* Example: Replace the word "bananas":

txt = "I like bananas"

x = txt**.replace**("bananas", "apples")

**print**(x)

* Definition: The ***replace()*** method replaces a specified phrase with another specified phrase. All occurrences of the specified phrase will be replaced, if nothing else is specified. Syntax:

string.**replace**(oldvalue, newvalue, count)

* ***oldvalue*** Required. The string to search for
* ***newvalue*** Required. The string to replace the old value with
* ***count*** Optional. A number specifying how many occurrences of the old value you want to replace. Default is all occurrences

txt = "one one was a race horse, two two was one too."

x = txt**.replace**("one", "three", 2)

**print**(x)

**24.8 Python String *strip()* Method**

* Definition: The ***strip()*** method removes any leading (spaces at the beginning) and trailing (spaces at the end) characters (space is the default leading character to remove). Syntax:

string.**strip**(characters)

* ***characters*** Optional. A set of characters to remove as leading/trailing characters
* Example: Following, removes spaces at the beginning and at the end of the string:

txt = "     banana     "

x = txt**.strip**()

**print**("of all fruits", x, "is my favorite")

* Example: Remove the leading and trailing characters:

txt = ",,,,,rrttgg.....banana....rrr"

x = txt**.strip**(",.grt")

**print**(x)

Redefined: Practice

PLACEHLDR = "[name]"

#*reading names and store it as a list*

**with** **open**("./Input/Names/invited\_names.txt") **as** invited\_peple\_file:

    names = invited\_peple\_file**.readlines**() #*returns the whole doc as list of lines with "\n"*

**with** **open**("./Input/Letters/starting\_letter.docx") **as** letter\_unedited:

    content = letter\_unedited**.read**() #*returns the whole doc as a string*

    #*stripping a name and insert it in the content*

**for** name **in** names:

        stripped\_name = name**.strip**() **#*"\n" is considered as space so no need to specify***

        letter\_content = content**.replace**(PLACEHLDR, stripped\_name)

**with** **open**(file = f"./Output/ReadyToSend/send\_to\_{stripped\_name}.docx", mode= "w") **as** edit\_letter:

            edit\_letter**.write**(letter\_content)

#*py main\_Copy.py*