Programming Club Meeting 13 Slides

Review

Modules

Basics

- AKA library
- Set of premade functions / variables that can be used in your code

exModule.py

reviewMeeting13 > exModule.py >

```
1  # Named Constants
2  num1 = 12
```

```
# Keywords
from exModule import num1
print(num1)
import math as m
print(m.pi)
print(dir(m))
```

```
12
3.141592653589793
['__doc__', '__file__', '__loa
n', 'atan2', 'atanh', 'ceil',
m1', 'fabs', 'factorial', 'flo
'isinf', 'isnan', 'isqrt', 'ld
'prod', 'radians', 'remainder',
bar
```

Custom Modules

- Have a Python file with functions / variables in it
- Use import to “copy” all of the code over
- Dot naming scheme fileName.funcOrVarName

exModule.py

reviewMeeting13 > exModule.py >

```
1   # Named Constants
2   num1 = 12
3   num2 = 45
4
5   # Ex Function
6   def foo() -> str:
7       return "bar"
```

```
# Example Custom Module
import exModule
print(exModule.foo())
print(exModule.num2)
```

```
bar
45
```

Time

```
# Time Module
import time
print(time.perf_counter())
time.sleep(2)
print(time.time())
```

v Output

```
0.018457993
1673483247.871295
```

Math

Code

```
1 # Math Module
2 import math
3 print(math.ceil(0.4))
4 print(math.fabs(-83))
5 print(math.floor(0.8))
6 print(math.sqrt(16))
7 print(math.pi)
8 print(math.inf)
```

Output

```
1
83.0
0
4.0
3.141592653589793
inf
```

Random

Code

```
1 # Random Module
2 import random
3 random.seed(3)
4 print(random.randint(0, 99))
5 options = ["a", "b", "c"]
6 print(random.choice(options))
7 random.shuffle(options)
8 print(options)
9 print(random.random())
```

Output

```
55
c
['c', 'b', 'a']
0.8929469543476547
```

Binary Search

Searching

- To find an item in a list, the easiest way is to look through each item until you find it or don’t, but this is rather slow
- If the list is sorted, we can use binary search
- Instead of going 1 by 1, the computer uses a divide and conquer method

Implementing Binary Search

- Find the midpoint of the list and determine if that value is the target, too high, or too low
  - The “target” is that you’re searching for
- If the item is too high, look only at the items before that value
- If too it’s too low, only look at the items after it
- If it is the value, you’re done
- Repeat
- If the computer runs out of values to look at, the target isn’t in the list

Visual Example

Example

Code

```python
def binarySearch(lst: list, target: int) -> int:
    """
    Returns the index of the target if present, otherwise
    returns -1.
    """
    low = 0
    high = len(lst) - 1
    mid = 0

    while low <= high:
        mid = (high + low) // 2

        if lst[mid] < target: # ignore left
            low = mid + 1
        elif lst[mid] > target: # ignore right
            high = mid - 1
        else: # found target
            return mid

    return -1 # target not in list

lst = [2, 3, 4, 10, 40]
target = 10
print(f"{target} is at index {binarySearch(lst, target)}.")
```

Output

```
10 is at index 3.
```

Files

Opening a File

- Modes:
  - “r” - read, default, error if file doesn’t exist
  - “a” - append, creates file if it doesn’t exist
  - “w” - write, creates file if it doesn’t exist
  - “x” - create, error if file already exists
  - “+” - include with “r” or “w” for read and write permissions

```
# Picking Mode + Read
with open(fileName, "r") as f1:
    data1 = f1.read()
    print(data1)
```

```
Hello! Welcome to file.txt
This file is for testing purposes.
Good Luck!
```

File Methods

- Read
- Readline
  - Can also be achieved with a for loop
- Readlines
- Seek
- Write

Practice

Practice Problem 1:

Word Frequency

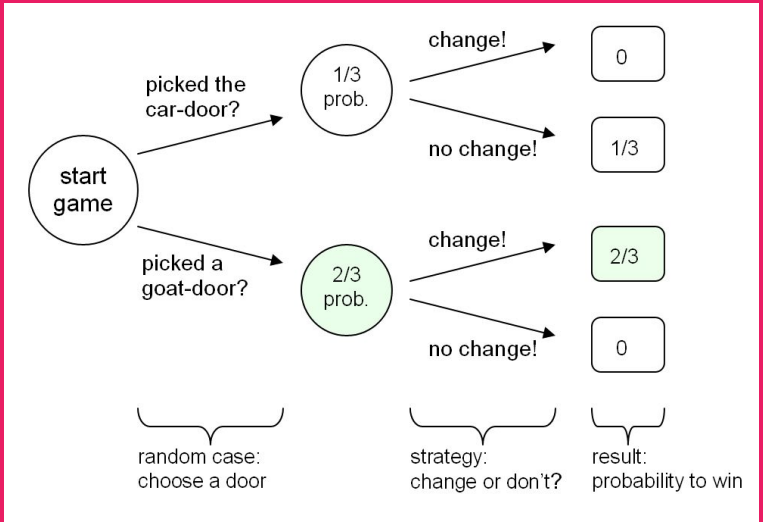
- Src: https://www.w3resource.com/python-exercises/file/python-io-exercise-10.php
- Goal: Write a Python program that will determine the number of times that each word in the file occurs.
- Relevant Information:
  - You can assume that there are only words, spaces, and newlines in the file
  - Words should be counted as the same even if the capitalization is different
  - You can find the text file for this problem on the GitHub in the “wordFrequency” folder

Practice Problem 2: Missing Index

- Goal: Write a Python program that will determine the index value of the missing entry in the text file.
- Relevant Information:
  - You should look for lines that have an index on them and determine if that is the correct next value
  - You can find the text file for this problem on the GitHub in the “missingIndex” folder

Practice Problem 3:

Monty Hall Problem Simulation



- Goal: Write a Python program that will simulate the Monty Hall problem with 10,000 runs.
- Relevant Information:
  - The Monty Hall problem involves 3 doors, behind 2 are goats and behind the other is a car. After a game show contestant picks a door, the host will remove one of the doors with a goat behind it (they will not remove the door that the contestant picked). The contestant can then choose whether to switch to the other remaining door or keep their choice. The problem asks whether the better choice is to switch or stay.
  - You should run the problem with randomized doors 100,000 times with and without switching and output the number of correct guesses for each method

Next Meeting: More Algorithms