Unit - 2

Lists

- It's basically an array.
- How to make a list?

1. Constructor | eg. `l = list()`
2. Hard Code | eg. `l = []`

- Can be Homogenous or Heterogenous.
- 0 Indexed.
- Mutable.
- Occupies a fixed memory location. Memory location does not change on update, unlike a string or integer.

```python
>>> l = [1,2,3,4]
>>> id(l)
4493687552
>>> l.append(5)
>>> id(l)
4493687552
>>> i = 5
>>> id(i)
4492249520
>>> i = 55
>>> id(i)
4492439600
```

- We can use a for loop to iterate through a list.

Weird thing - List copying:

```python
>>> l = [1,2,3,4]
>>> l_copy = l.copy() #.copy does a proper copy
>>> l_copy
[1, 2, 3, 4]
>>> l.append(5)
>>> l
[1, 2, 3, 4, 5]
>>> l_copy
[1, 2, 3, 4]
>>> l_copy1 = l # this just appoints the same memory location
>>> l.append(6)
>>> l_copy1
[1, 2, 3, 4, 5, 6] # Every change in l is reflected in l_copy1
```

Functions that a `list` supports:

```python
list.insert(<position>, <element>)
list.append(<element>)
list.sort()
list.pop(<index>)
list.remove(<element>)
list.count(<element>)
```

Tuples

- they are weird
- basically list but it's immutable
- No clue why they exist
- supports concatenation & multiplication
- BUT, you can modify the data inside a tuple. Say for example, there is a list inside a tuple, you can modify the list. Don't ask me why
- Some examples:

```python
>>> a = 'aeiou'
>>> tu = tuple(a)
>>> tu
('a', 'e', 'i', 'o', 'u')
>>> tu1 = (1,2,[1,2,3])
>>> tu1[2][1] = 3
>>> tu1
(1, 2, [1, 3, 3])
>>> whyyyy
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'whyyyy' is not defined
>>> tu2 = tu1 + tu
>>> tu2
(1, 2, [1, 3, 3], 'a', 'e', 'i', 'o', 'u')
>>> tu2 * 2
(1, 2, [1, 3, 3], 'a', 'e', 'i', 'o', 'u', 1, 2, [1, 3, 3], 'a', 'e', 'i', 'o',
'u')
```

Dictionary

- Key Value Pairs.
- Also Called a Map.
- **Unordered ONLY UNTIL PYTHON 3.7 | After 3.7, they are hash tables.**
- How to initialise?

```python
dictionary = {}
dictionary = dict([(key1, value1),(key2,value2)..etc]) # nested list of tuples
```

- Has constant(O(1)) lookup times.
- Methods

1. .keys() => returns an iterable object of keys.
2. .values() => returns an iterable object of values.
3. .items() => returns tuple of key value pairs in a list.
4. .clear() => makes the dictionary empty
5. .pop(`required key`) => returns popped element and pops key:value pair.
6. .popitem() => randomly removes a key:value pair.
7. .fromkeys(`iterable object, value`) => creating keys from iterable object

Q. Write a python program to create a dictionary with the SRN, Name, and Marks Of 'N' Students. Display the names of students above 50.
Q. Write a python program to find the highest 2 values in the dictionary.

Functions

- Block of statements which does a job.
- Has a name.
- Why function? To not repeat a piece of code again and again. **DRY: Don't Repeat Yourself.**
- Advantages
  - Easy To Maintain
  - Improves Modularity
  - Easy testing and debugging
  - Reusability
  - Readability
- Function header:

```python
#definition
def functionName(parameters):
    #statements

#function call
functionName(arguments)
```

> **Function** vs **Method:**
> A *function* is accessible from the global scope, Examples include `min()`, `max()` etc.
> A *Method* is a function that is accessible only by referencing an object. These functions are specific to that object only. Examples include `list.copy()`, `list.pop()`, `list.remove()` etc.

Q. Write a python program to determine if a number is even or odd using a function.

Argument vs *Parameter*

1. **Arguments** : values passed during function call

2. **parameters**: values passed during function definition

> An ***Activation Record*** (AR) is **a private block of memory associated with an invocation of a procedure**. It is a runtime structure used to manage a procedure call. An AR is used to map a set of arguments, or parameters, from the caller's name space to the callee's name space.

Function Signature

- The name and the number of arguments together are called a function signature. When the number of variables during function call does not match, we say that the *function signature* does not match.

> When we return multiple values in a function, it returns as an *unnamed tuple*

Sets

- *Unordered* collection of *unique elements*.
- Syntax:

```python
s = set(<optional list of elements>)
#or
s = {1,2,3}
```

- Elements are *unique*.
- Can be *heterogeneous*.
- Elements *Must Be Hashable*:
  - Hashing is a mechanism to convert the given element to an integer
  - An object is hashable if it has a hash value which never changes during its lifetime, aka *immutable* (it needs a `__hash__()` method).
  - Non hashable examples: list, dictionary, etc.
- No *Indexing*
- A set *is mutable*.

> If you want to use `min()` or `max()` on a set, the datatypes must be compatible i.e., they should be homogenous.

Methods of sets:

```python
>>> s1 = {1,2,3,4}
>>> s2 = {3,4,5,6,7,8}
>>> s1.union(s2)
{1, 2, 3, 4, 5, 6, 7, 8}
>>> s1
```

```
{1, 2, 3, 4}
>>> s2
{3, 4, 5, 6, 7, 8}
>>> s1.difference(s2)
{1, 2}
>>> s2.difference(s1)
{8, 5, 6, 7}
>>> s1.symmetric_difference(s2)
{1, 2, 5, 6, 7, 8}
>>> s2.symmetric_difference(s1)
{1, 2, 5, 6, 7, 8}
>>> s1.remove(1)
>>> s1
{2, 3, 4}
>>> s1.discard(5) # will not throw an error even if element not in set
>>> s1.pop()
2
>>> s1
{3, 4}
>>> # randomly pops an element
>>> del s1
>>> s1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 's1' is not defined
```

Command Line Arguments

- Arguments given while a command is run.
- Example:

```
python setup.py --argument
```

How do we implement these in our code?

```python
import sys

# total arguments
n = len(sys.argv)
print("Total arguments passed:", n)

# Arguments passed
print("\nName of Python script:", sys.argv[0])

print("\nArguments passed:", end = " ")
for i in range(1, n):
    print(sys.argv[i], end = " ")

# Addition of numbers
```

```
Sum = 0
# Using argparse module
for i in range(1, n):
    Sum += int(sys.argv[i])

print("\n\nResult:", Sum)
```

Important Shit:

- The default return type of `sys.argv` is a list of strings.
- The first element i.e, index[0] is the name of the python file itself
- The rest of the arguments are the actual arguments passed.

Files

Code	Function	Special Note
r	read	
w	write	overwrites even if you do not perform any write. Creates if does not exist.
a	append	appends
r+	read+write	reads and writes. overwrites only if written to
w+	write+read	overwrites even if there isn't any writing. Creates if does not exist. Allows read too.
a+	append+read	append and read. Creates if does not exist. Best in my opinion.
rb	read binary	
wb	write binary	
ab	append binary	
rb+	read+write binary	
wb+	write+read binary	
ab+	append+read binary	

Text Stuff:

```python
# performs auto close
with open("test.txt", 'r') as f:
    print(f.readline()) # reads one line only
    print(f.readlines()) # returns list of all lines including escape sequences

# Writing
with open('test.txt', 'a') as f:
    f.writeline('This is a test line')
```

Q. Write a python program to print all the lines in a file using `readline()`
Q. Write a python program to return the frequency of a word 'python' in a text file.

> ℹ️Refer my GitHub for all the answers to the questions in this pdf. Find all the code inside the /class_stuff directory. Link: [Click Here]

Made with ♡ by Anurag Rao