Programming 1 - python Notes

Created	@November 9, 2023 10:46 PM
Type	Sapienza
Reviewed	☑️

▼ Important Functions

- **enumerate** - creates a iterator containing tupples making the first containing the index and corresponding value in the list
  - you can convert it to list

```
enumerate(a_list)
```

- **len - returns** length of itterables

```
len(...)
```

- **map** - to apply a single fun to every elmt of iterable returns an iterator so you need to change it to a list or something else

```
map(a_fun, iterable)
```

- **print**

```
print(objects, sep='any char to separate them', end='chars
```

- **sorted** - returns the sorted copy of a list but doesn't change the original list

    - key - any function that can be used to compare elements which could be our own function.
        - how it works is : evaluate the keys → sort the keys → output the sorted values
        - if we add '-' sign before the function, it will be reversed
        - key=(fun1, fun2) - first based on fun1 then fun2

```
def a_fun(a)
    return #an elmt based on w/c the list is sorted
sorted(a_list, key=a_fun, reversed=True/False)

# example
sorted(a_list, key=len)
```

- **lambda function** - used when we want write simple function without a name usually one line

```
# Example
sorted(a_list, key=lambda x:(len(x), x[0]))
# given an argument x returns len(x) and x{0}
```

- **Unicode functions**

```
# get unicode positions
ord('char')

# get character of a unicode
chr(unicode)
```

- **Type conversion**

```
# string, floats -> int
int(...)
```

```
# strings, sets -> list
list(...)

# int, string -> float
float(...)

# list, string -> set
set(...)

# any type -> string
str(...)
```

- **dir() - to get available functions and methods of data type, module**

```
dir(...)
```

▼ OS library

- to work on directories
- imported as `import os`
- to create a directory - `os.makedirs('name or path of dir', exist_ok=True)`
- To check existence - `os.path.isdir('folder_dir')` - returns a Boolean
- To list of all objects in a directory **`os.scandir([path])`**
  - returns an iterator so we need to change it to list
  - if we want just the names use `item.name` method while converting it to list
- To get the destination in a path **`os.path.basename(path)`**

▼ Recursion

- A function that calls itself and uses the value returned by itself .
- **Principles**

- **Base Case (**identify the simplest possible inputs it can solve) - the condition to stop the recursion
- **Simplify** (in every function call it should be able to simplify problem)
- Pass the result of most inner function to the previous function call

▼ Built in Modules

```
# to import
from [module name] import [fun name]

# or import the whole module
import [module name]
```

▼ Collections

▼ Files

- a text file is iterable (in to lines)
- for reading a file - file_ref = open("file path", 'r', encoding='utf8') - the file path could be relative or absolute
- for writting a file. - file_ref = open('file path" , 'w', encoding='utf8') -
  - if the file doesn't exist it will create
- file_ref = close() - for closing
- if we use r,w,a arguments python reads it as text. if br, bw,ba as binary
- file_ref.readline() - to read the first line. if it is called again it reads the next line
- .readlines()
- .write("what ever we want") - to write text in a file
  - must be done after opening a file for writing
- print(what_ever_we_want, fileRef) - has the same result as write

▼ JSON FILES

```
# reading the file
import json
with open(file_path) as var1:
    var2 = var1.read()

# converting raw string into a dictionary
parsd_json + json.loads(var2)
```

▼ Images

- every pixel takes 24 bites (R =8, G=8, B=8)
- In RGB if R=G=B the the color is either black, gray or white
- to create image

```
import images

matrix = [...]
images.save(matrix, 'path')

# or

import pngmatrix
var_name = []
#append the rgb of every pixel as a matrix the each raw be
pngmatrix.save_png8(image, “file_name.png”)
```

- to read image

```
var_name = pngmatrix.load_png8(filename) - it assign a
```

▼ Lists

- To create new list `list_a = []` or `list_a = list()`

- **Accessing items**

```
# single elmt at index i
a[i]

# range from index i to j
a[i:j]  # if either or both left use index 0 or last

# range from index i to j in s steps
a[i:j:s] # s can be -ve
f
```

- List comprehension

```
[i**2 for i in range(9)]
{i:i**2 for i in range(5)}
[(x,y) for x in range (19) for y in range {13) if (x+y)
```

- **Deep copy -** after copying a list using .copy method if we change the copy it will affect the original because the copy contains just the reference to the location. to solve this

```
import copy
new_copy = copy.deepcopy(original_copy)
```

- **Methods**

```
# conctatenate a list of strings
''.join(a_list_of_str)
```

▼ Strings

- To create a empty string `var_name = ''`
- does not support item assignment
- **Formatted String**

```
# everything in {} is treated as normal python and the r
f'string {5/2} is {var_name}'
```

- **Methods**

```
# remove the first and last white space
.strip()

# checks if it contains only alphabetic char
.isalpha()

# checks if it contains only number char
.isdigit()

# replace char 'a' with char 'b'
.replace('a', 'A')

# change to Upper or lower case cha
.upper()
.lower()
```

▼ Tuple

- If you are creating a tuple with just 1 element

```
a = (5,) # because we don't want the bracket to be treat
```

- unpacking and packing
- a tuple can be used as key in dictionary unlike lists

▼ Dictionaries

- are mutable
- .keys(), .value(), .items()

▼ Sets

- .discard() - to remove from a set (it doesn't complain if the elmt doesn't exist unlike .remove())

▼ Function

- how to receive of only a specified type as an arguement

▼ Classes

- is like a blueprint or template that define the characteristics and behavior of an object
- Are user defined data types
- can contain Method and attributes
  - **Attributes** - variables to store info of an objects
- class methods should contain a `self` parameter
- if you want some attributes to be mandatory put them as a parameter in the `init` method
- `__init__` is a constructor method which implicitly is called when ever you create a new object

```
# syntax
class <class_name>:
        ,<initialize_class_attributes_and_methods>
# to use it
<var_name> = class_name()
class_name.attribute_name = <some_thing>

# example
class employee:
        def __init__(self):
```

```
                self.ID = None
                self.name = None
                self.age = None
        def can_retire(self):
                if self.age > 65:
                    return True
                return False
E1  = employee()
E1.name = 'Abenezer'
E1.age = 22
E1.Id = '2114802'
E1.can_retire() -> False  # we don pass an arguement b/c E1 b
```

▼ Tree

- Most of the time we create it using classes
- **Root -** the first node in a tree
- **Leaves -** node with no child
- **Internal trees -** a node having at least 1 child

▼ Binary Tree

A data structure in w/c a data is linked to two successors left branch and right branch

- **Pre-order visit -** print → Left → Right
- **In-order visit -** Left → Print → Right
- **Post-order visit -** Left → Right → Print

```
class Tree(object):
    def __init__(self):
        self.left = None
        self.right = None
        self.data = None

root = Tree()
```

```
root.data = "root"
root.left = Tree()
root.left.data = "left"
root.right = Tree()
root.right.data = "right"

print(root.left.data)
```

▼ N-ARY tree

- each data can be linked to multiple child nodes
- EG - Folder structure is also a tree

```
class Tree(object):

    def __init__(self):
        self.left = None
        self.child = []
        self.data = []

    def createChildren(self,amount):
        for i in range(0,amount):
            self.child.append(Tree())

    def setChildrenValues(self,list):
        for i in range(0,len(list)):
            self.data.append(list[i])

root = Tree()
root.createChildren(3)
root.setChildrenValues([5,6,7])
root.child[0].createChildren(2)
root.child[0].setChildrenValues([1,2])
# print some values in the tree
```

```
print(root.data)
print(root.child[0].data[0])
```

▼ Methods

are functions that can be called on objects

```
# to create a copy of something
.copy()
```

▼ Loops

- **While -** executes a set of statements multiple times as long as the condition is True

```
# sythax
while [conditions]:
    [statements]
```

- **for -** executes a set of characters for each element of an object

```
# syntax
for [var] in [iterable] # var refers to each element of th
    [statements]
```

▼ Keywords

- **del** - to delete a given object

```
del list_a[3]
del var_name
```

- **pass -** to avoid error if a function doesn't contain anything
- **and -** returns True if both are True, if not False

    - if the first statement is false it stops checking
- **or -** returns True if either of the statements is True
- **continue** - stops to execute the current iteration and continues to the next in a loop
- **in** - to check if some item/string in iterable
- **“_”** - represents the last returned object which is stored in memory

```
some_variable = _
```

▼ **To cover**

- what are generators?
- spyder - %timeit
- exceptions (try except)
- Spyder shortcuts
    - f9 - to execute only the selected code
    - ctrl + shfrt + I/E - naviga b/n interpreter and ediror
- Py format Python String format() Method (w3schools.com)
- with open as f
- f.writelines(...)