TEST CSE101 #1

WEEK 1:

Ints : (-1, 0, 1, 2, …)
Floats : (3.14, 2.773e-10) → **only approximations to real numbers - be careful when comparing**
Operations : +, -, *, /, // (floor division, euclidean division quotient), % (modulus) , ** (exponential)
Keywords → reserved words (33) - you can not use them as names
- Special keywords : False, None, True

COMPARISON : returns True or False
- x == y (equality)=> used to compare two elements
- X = y (to assign something to a variable)
- x != y (inequality)
- x > y and x < y (strict inequality)
- x <= y and x >= y (inequality or equality)

If there is no return function in code, then None will be returned by default → every function must return something even if it is None
IF: we can combine more than one condition using OR, AND, NOT

WEEK 2:

For loops: i starts at 0 so final value is your stop value-1
Boolean values - True and False
- Compare string to string and int to int...

Lists:
- Square brackets []
- Sequential data
- We can put anything we want in a list - mixing elements — can put None, True, False
- Index of a list starts at 0
- Use negative indices to start from the end of the list
- Can add lists: big_list = list1 + list2 + …
- *len(list)* = length of the list (number of indices)
- Add an element to the list: use append: *list.append(element)*
  If you append a string use '': *list.append('element')*
  If you append en number don't use '': *list.append(number)*
  => add the element at the end of the list
- Add an element at a precise index: use insert: *list.insert(index, element)*
- Check if an element is in the list: *element in/not in list*

If in the list (otherwise error):
  - Find the index of its first occurence: *list.index(element)*
  - Remove the element: *list.remove(element)*
  - Replace elements: list[index] = new element
- Using for loops: iterate over the list: *for i in list:*
- Empty list: *list = []*
- Can define use a for loop in a list: *list2= [f(x) for x in bla]*
- min(list) = minimum value of the list
- Idem for maximum: max(list)

Tuples:
- Used to pack multiple values into a single object
- Immutable - once created you can't replace the values - can't add, remove, lengthen or shorten
  But you can do it for if an element of the tuple is a list:
  Ex:
```
t = ('letters', ['a',' b'])
t[1] = ['a', 'b']
t[1].append('c')
t[1] = ['a', 'b', 'c']
t = ('letters', ['a','b', 'c'])
```
- Ex:
  - date = (13,'February',1982)
    day = date[0]
    month = date[1]
    year = date[2]
  - Easier: (day, month, year) = date
- empty tuple is given by empty parentheses → a = ()
- Index starts at 0

WEEK 3:

STRINGS
- A lot of operators and methods work for strings and lists but can't modify a string
- 'x' in 'Excellent' → returns true
- string[index] == letter or symbol
- .index()
- len()
- sentence.split() → splits by space (separate each word)
- sentence.split('; ') → splits by what it in parenthese
- Can't do string[2] for a string
- separator.join(list_of_strings)
  whatwillsepareeachelementofthelist.join(list)

- words = ['get', 'on', 'up']
  phrase = ' '.join(words)
  phrase == 'get on up'

SLICING:

- To extract a part of list/string:
- list[start:end]
  string[start:end]
  By default start is 0 stop is the end of the list/string
  [:6] => starts from the beginning to index 6 (not included)
  [3:] => starts at index 3 goes to the end (included)
  WARNING: the ending index is not included
- Can add strpping: name[start_from:stop_before:stepping]
  Ex: x[1:4:1] == ['b','c','d']

FILE OPERATIONS:

- Open a file if you want to :
  - read it: with open('input-filename','r') as infile:
    Then use: s = infile.read() to read whatever you want in it and assign what you read to s (to use it afterwards)
  - It's 'r' by default (if you omit the 'r'):
    with open('sample.txt') as infile: #will read the file
  - Write in it: with open('output-filename','w') as outfile:
    Then use outfile.write(something) to write in the file
    '\n' => new line
    '\t' => tabulation
    ' ' => white space
- For loops in read: for line in sample: do smthng with line
- If you want to do more than one loop with a file you have to re open the file for each loop

WEEK 4:

Dictionaries : collections of key → value pairs
They map keys to values.

Keys	Values
Unique hashable(typically immutable) Ex:strings, numbers, tuples	Can be repeated

{key_1 : val_1, …, key_n : val_n}
Empty dictionary → {}
Testing membership : in
- In [1]: d = {'Jan':1,'Feb':2,'Mar':3,'Apr':4}
  In [2]: 'Jan' in d
  Out[2]: True
  In [3]: 'May' in d
  Out[3]: False

To remove a (key: value) pair : del ex: del phone_book['Alessandro']
To add pairs: dictionary['key']=value
Modify a value: reinsert the value: dic[key] = new value
To iterate over :
- the keys of a dictionary d : for k in d.keys():
- the values : for v in d.values():
- Key:value pairs as tuples : for (u,v) in d.items():

Create dictionaries name = {w[0]:w for w in words)}
- words = ['Alpha', 'Bravo', 'Charlie', 'Delta', 'Echo']
  nato = { w[0]: w for w in words }
  nato['A'] == 'Alpha' # True
  nato['C'] == 'Charlie' # True

Concatenate : +
len(dictionary)

ERRORS:

- Syntax error → the program is not well-formed - piece of text does not represent part of a legal python program
- KeyError: writing everything you have in a dictionary in a text but you forget what's in the text which is not in the dictionary
- Runtime error → non-existent index
  - ValueError
  - AttributeError: runtime error associated with a bad attempt to use some attribute of an object. Ex: using a function on a list

CODE - FUNCTIONS:

def function_name(arg1, ..., argM)
int(x)
float(x)
str(x)
print(x) → x has to be a string

intput(prompt)

- In [1]: who = input('Please enter your name: ')
  Please enter your name:
  Python is now listening for keyboard input (until you hit 'Enter')

pass - let a function do nothing (maybe temporarily)
if condition :
else :
elif condition :
while condition :
Continue : skip the rest of the loop iteration and go to the next one
and, or, and not
for i in range(stop):

- ***Warning:*** *always starts from i = 0; does not include i = stop.*

for i in range(start, stop, stepping):

- ***Warning:*** *loop includes i = start, but not i ≥ stop*

LIST:

list[x] - x can be any number (negative start from end and 0 + positive from beginning) - if number doesn't exist → runtime error
len(list)
list = [‘’, ‘’, ‘’]
list.append(element)
list.insert(index,element)
in, not in (element in list)
list.index(element) → first occurence of element
list.remove(element) → removes first occurrence
continue - skip rest of iteration in loop → onto the next iteration
divmod

- quotient, remainder = divmod(1234567, 121) # Now quotient == 1234567 // 121 # And remainder == 1234567 % 121

FILES:

with open(file_name,'mode') as file:

- Mode: ‘w’, ‘r’, ‘a’ (write, read, append)

outfile.write(something) → write to file
s = infile.read() → read from file
for line in file:
    c = len(line)
.strip() → remove unwanted white space (‘ ’, ‘\n’, ‘\t’)at the beginning and end of str
.strip(‘!?’) or strip(‘<’) remove !? or <
.strip(str) → remove other leading and trailing characters

string[0] → gives first letter of string (acts the same way as list)

SLICING:

val_name[start_from:stop_before] → for list and str

- Negative numbers work. Start from end. -1 is the last value

name[start_from:stop_before:stepping]

s.format() → '{}{}:{}({})'.format(p,d,q,l) *it's a string*

DICTIONARY:

del dictionary['Alessandro']

- 'Alessandro' in dictionary # False

Item → (key:values) d-> dictionary

```
for k in d.keys():
for v in d.values():
for (u,v) in d.items():
```

TEST 2

CLASSES :

- class names start with CAPITAL letter: class **T**hingo:
- create new objects we call their class name like a function: **x = Thingo()**
- *type(object) -> gives the object class*
- To call attributes of obj: ***object.attribute_in _question***
- To call an object's method: ***object.method(attributes_needed)***
- *def __init__(self, attributes): d*efine how to initialize these objects
- *def method(self, variables):*
- your class's __str__ method tells the builtin function str how to compute string representations of the objects in that class. Use ***str(object)*** NOT *object.__str()__*
- repr(x) is meant to produce a "canonical" or "official"string representation of an object x. Typically we use repr(x) to produce something useful for debugging.

repr(x) => debugging, programmers (equivalent to x, to the object)
str(x) => user

```
In [1]: b = Birthday('James',23,5,1993)

In [2]: str(b)

Out[2]: 'James: born May 23 (1993)'

In [3]: print(b) # equivalent to print(str(b))

James: born May 23 (1993)
```

```
In [4]: repr(b)

Out[4]: 'Birthday(James,23,5,1993)'
```

- __eq__ method tells Python how to test if two objects are semantically equal:
  def __eq__(self, variable):

COLLECTIONS:
Collections are objects that aggregate other objects: lists, dictionaries, sets

HASH:
A hash function maps objects to integer *hash values*.
In DICTS:

- hash: maps key objects to integer hash-values, which are then used to index entries in the array. hash(key) gives us a shortcut to the cell where (key,value) should be.

–> To use it we also need an __eq__ method

To define __hash__ method:

- Use **data attributes that don't change over time**
- DON'T use class attributes nor attributes that might evolve
- Hash link the object to an integer which is the position of its key in the array (dictionary)
- TO HASH: the simplest way is to define the hash as a simple linear combination of the `hash(a)` for each fixed data attribute `a` in the object

SETS:

- **unordered** collections, without repetition of elements.
- create them using curly brackets: {a,b,...}. Or create empty set: `set()`
- can also construct sets directly from iterable objects: `q = set(range(10))`
- You can think of a set as being a like dict with:
  Set elements corresponding to dict keys, and
  No values
- Delete one element: `set.discard(element)`
- `set.add(element)`
- `set.clear()` delete all its elements

CODE FUNCTIONS:
Files:

- *`name_of_file.readlines()`* -> gives a list of all the lines in file

.strip() -> srips everything: ' ' '\n' '\t'

MORE COMPACT METHODS:
List = [name for name in file]
range(int) or range(int1,int2)
list.pop() : delete and return last element of the list
list.pop(i): idem but object at index i
random.shuffle(list)
'The winners are ' + ' and '.join(win for win in winners)