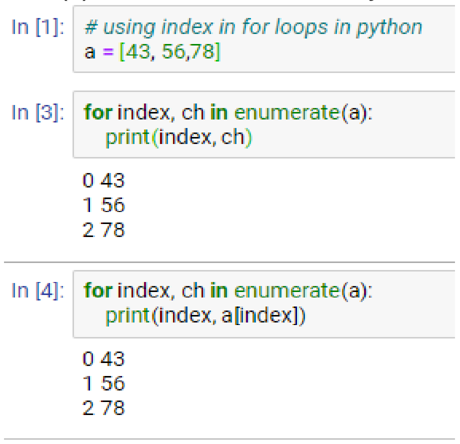
PYTHON NOTES - W1

- Types of programming languages:
    1. **Procedural:** procedures like if/then,while,switch,function(flow of program)[example-C,Java], Disadvantage- data hiding is not available.
    2. **OOP** -- can divide program in blocks, concept of classes [Example - C++, **Java,Python(**both are hybrid language**)**],
       Principal of OOPs
        - Inheritance
        - Abstraction
        - Encapsulation
        - Polymorphism
    3. **Functional** programming: **functions are treated as 1st class citizens** [Example - Python].
        - **First-class functions** when functions in that language are treated like any other variable. For example, in such a language, a function can be passed as an argument to other functions, can be returned by another function and can be assigned as a value to a variable.
- **Primitives:** int,bool,float,char. **All primitives in Python are objects, no such concept of primitives.**
- **Class** is a combination (bundle) of Properties **(variables) + Methods (functions),** instance of class is called **Object.**
- **Object** is a global class, and every other class inherits the object class.
- **Dynamic vs Static Language**: if int is provided it will keep it as it is in Java, while in Python type of variable is determined during runtime. During compilation, type_check is performed.
  int a = 10; // LHS happens at compile time whereas RHS happens at runtime
- Types of memory in any programming language:
    1. **Stack memory**:
        - reference variable is saved
    2. **Heap memory**
        - Object is saved
        - Actual object value is saved somewhere in RAM
          `a = 10` // a is a reference which resides in stack memory whereas 10 is an object residing in heap memory.
- More than one reference can point to an object, but a single reference variable cannot point to 2 objects simultaneously.
  Garbage collection eats up unreferenced object from memory
- **== Compares values whereas "is" compares references**.
  -6 to 256 are reserved by python, i.e.

```
a=10
b=10
print(a is b) → true (same object(same reference))
a= 4555
b= 4555
```

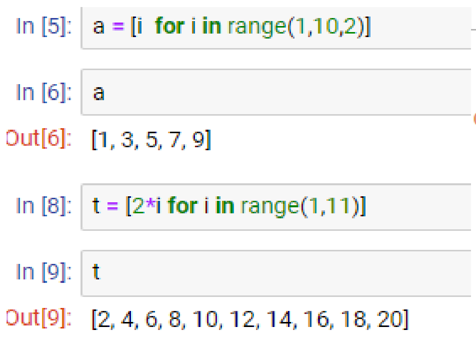
```
print(a is b) → false(coz, -6 to 256 is reserved, and numbers beyond
this range will have different objects(different reference), even for
same values).
```

- Size of integer in Python: RAM in the computer
- Python converts all int to float before performing division.
- Python is a strongly typed language as it **does type check** of all variables during runtime, loosely typed language(e.g. - C).
- // gives floor division(integer value)
- NameError(if interpreter doesn't recognize something; sees a variable which was never defined) & SyntaxError(if there is something wrong in the way a program should be written) are 2 errors in Python.
- Data types: **Mutable(can change value) & Non-Mutable(cannot be changed)**
- Array is homogeneous (elements having the same type of data type) whereas list is heterogeneous.
- List is built either using [] or using list function. (list()).
- Tuple is immutable---value of object cannot be changed, use () for making tuple.
- Set (unique collection of values) is unordered, non-duplicate values
- Dictionary-- key value pair, key should be unique.
- Reverse a list a[-1::-1]
- Dictionary is a key-value pair database.
    1. dict() is the constructor used for manually making a dictionary.
    2. Dictionaries have no order.
    3. The get() method returns the value of the item with the specified key.
    4. *dictionary*.get(*keyname, value*)
    5. Using the "for" loop, it loops on keys not on values.
- Mutable -- list[], set, dict{key:value}
- Immutable -- tuple(), string
- Tuples are immutable.
    1. Tuples are comparable
- Zero(0) stands for false and any other number stands true, empty list [] is also false.

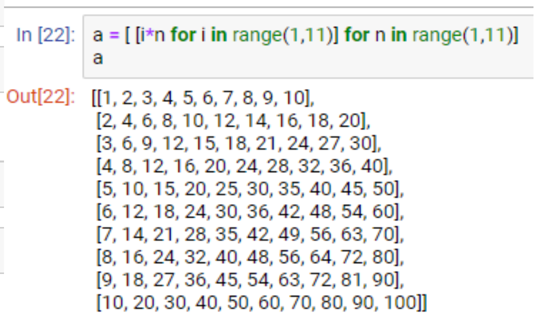


```
In [1]: # using index in for loops in python
        a = [43, 56,78]

In [3]: for index, ch in enumerate(a):
            print(index, ch)

        0 43
        1 56
        2 78

In [4]: for index, ch in enumerate(a):
            print(index, a[index])

        0 43
        1 56
        2 78
```

- List comprehension



```
In [5]:  a = [i  for i in range(1,10,2)]

In [6]:  a

Out[6]:  [1, 3, 5, 7, 9]

In [8]:  t = [2*i for i in range(1,11)]

In [9]:  t

Out[9]:  [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```



```
In [22]:  a = [ [i*n for i in range(1,11)] for n in range(1,11)]
          a

Out[22]:  [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
           [2, 4, 6, 8, 10, 12, 14, 16, 18, 20],
           [3, 6, 9, 12, 15, 18, 21, 24, 27, 30],
           [4, 8, 12, 16, 20, 24, 28, 32, 36, 40],
           [5, 10, 15, 20, 25, 30, 35, 40, 45, 50],
           [6, 12, 18, 24, 30, 36, 42, 48, 54, 60],
           [7, 14, 21, 28, 35, 42, 49, 56, 63, 70],
           [8, 16, 24, 32, 40, 48, 56, 64, 72, 80],
           [9, 18, 27, 36, 45, 54, 63, 72, 81, 90],
           [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]]
```

- **Functions**
  1. If function has no return types, then function would return **None**
  2. Functions are also Objects!
  3. If your function body is small use **inline/lambda** function

     ```
     lambda argument : function body
     ```

     ```
     def add(a, b):
         return a + b
     OR
     add = lambda a,b: a + b
     add(3,4)
     7
     ```
  4. **map()** function returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable (list, tuple etc.)
     `map(fun, iter)`

     `Map?` Shows help on functions provided before ? (valid in jupyter notebook)

     ```
     a = [1,2,3,4,5]
     list(map(lambda x: x ** 2, a))
     [1, 4, 9, 16, 25]
     ```
  5. **filter()** method filters the given sequence with the help of a function that tests each element in the sequence to be **true or not**.
     `filter(function, sequence)`
     `Returns`: an iterator that is already filtered.It is normally used with Lambda functions to separate list, tuple, or sets.

     ```
     a = [1,2,3,4,5,6,7,8,9,10]
     list(filter(isEven, a))
     [2, 4, 6, 8, 10]
     ```
  6. In python, there is only 1 way to give comments i.e #
     multiline strings : """ """ or ''' ''', they are not comments

     ```
     def fun():
         """
         this is a comment(really?)
         """
     ```

```
def fun():
    # this is actually a comment
 File "<ipython-input-88-522eb4d9fa79>", line 2
    # this is actually a comment
                                ^
SyntaxError: unexpected EOF while parsing
```

7. `isinstance(obj, class_or_tuple, /)`
   Return whether an object is an instance of a class or of a subclass thereof.

   Classes are also object!

   ```
   def multiply(x): # x here is each element of a, not whole a!
       return 2 * x
   isinstance(multiply, object)
   True
   ```

- input() functions take everything as a string irrespective of the type you provide.
- The value between the parentheses when we call the function is referred to as an argument of the function call.(value passed while calling a function)
- **Positional arguments** because their assignments depend on their positions in the function call, **Keyword arguments**, where we explicitly refer to what each argument is assigned to in the function call.
- "**break**" keyword takes the cursor out of the while loop to the end whereas "**continue**" keyword skips the present iteration and goes to next iteration statement.