Introduction to Python

Dr. Sourav Kumar Dandapat

Comments in Python

- #This is a comment
- Comments can be placed at any position of a line, and Python will ignore the rest of the line
- print("Hello World") #This is a comment
- Multi-line comments:

```
print("Before comment")
"""
This is a
multi line
comment
"""
print("After comment")
Output: Before comment
After comment
```

See colab

Variable Naming

- There's no way to declare a variable without assigning it an initial value.
- Assigning a value to a variable itself declares and initializes the variable with that value
- Variables names must start with a letter or an underscore. The remainder of your variable name may consist of letters, numbers and underscores.
- Example:
  - abc=2
  - _xyz=3.5
  - A4x_5="I am a string"

Variable Naming

- You can not use python's keywords as a valid variable name
- Below is the list of python keywords
  - ['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
- Variable names are case sensitive.

Exercise

- ab_c=2
  - valid
- _abc=3
  - valid
- ab$1=5
  - Invalid, because it uses special symbol which is not allowed
- class=4
  - Invalid, because class is a reserve keyword
- 1sda=4
  - Invalid, because variable name can starts with either letter or underscore not digit
- sdA2=3
  - valid

Indentation

Block: A block is a piece of Python program text that is executed as a unit.

All instruction that need to be executed within a block must be properly indented, there is no delimiter of block in python using opening and closing braces or parenthesis like C

Example:

```
Condition to enter into a block:
        statement1
        statement2
        statement3
statement4
```

In the above example, statement 1, statement2 and statement3 are part of a block that is clear from indentation. However, statement4 is not part of that block

Indentation

Blocks that contain exactly one single-line statement may be put on the same line, though this form is generally not considered good style:

Condition to enter into a block: statement

Spaces vs. Tabs

- You may either use 4 spaces for indentation or a tab.
- Few style guide for Python code, states that spaces are preferred.
- Many python version disallow mixing of space and tabs.

Datatypes: Boolean

- **Built-in Types:**
  - **Booleans**
    - bool: A Boolean value can be either True or False. Logical operations like and, or, not can be performed on Booleans
  - Most Values are True
    - Almost any value is evaluated to True if it has some sort of content.
    - Any string is True, except empty strings.
    - Any number is True, except 0.
    - Any list, tuple, set, and dictionary are True, except empty ones.
    - Example
      - The following will return True:
      - bool("abc")
      - bool(123)
      - bool(["apple", "cherry", "banana"])

Datatypes:Boolean

- Some Values are False
  - empty values, such as (), [], {}, ""
  - number 0.
  - value None.
  - value False.
  - Example
    - bool(False)
      bool(None)
      bool(0)
      bool("")
      bool(())
      bool([])
      bool({})

Datatypes: Boolean

– Python Logical Operators

- x **or** y *# returns true if either of two operands is True*
- x **and** y *# returns True if both the operands are True*
- **not** x *# if x is True then returns False, otherwise True*

Datatype: Numbers

– **Numbers**
  - **int:** Integer number
  - a = 2
  - b = 100
  - Integers in Python are of arbitrary sizes.
  - **float:** Floating point number; precision depends on the implementation and system architecture
  - a=23456789.5
  - b=2.0
  - **complex**: Complex numbers
  - a = 2 + 1j

Datatype: string

- String can be represented using single quote or double quote
- For example "hello" or 'hello'
  - Assigning string to a variable: 'hello'
  - Assigning multi-line string to a variable
  - a = """This

    is a multi-line

    string."""
  - Instead of double quote we can also use single quote

Retrieving element(s)

- Strings are Arrays
- a = "Hello, World!“
- print(a[0]) #H
- Slicing- You can return a range of characters by using the slice syntax
- Get the characters from position 2 to position 5 (not included):
- a = "Hello, World!"
  print(a[2:5]) #llo
- Note: index starts from 0

Negative Indexing

- Use negative indexes to start the slice from the end of the string. -1 is the last character, -2 previous than last and so on
- a = "Hello, World!"
- print(a[-1]) #!
- print(a[-5:-2]) #orl

Operation on string

- String Length: len() function returns the length of a string
  - a = "Hello, World! "
  - print(len(a)) #13
- Removing white space from beginning or end using strip()
  - a = " Hello, World! "
  - print(a.strip()) # returns "Hello, World!"
- Make the string lower case using lower()
  - a = "Hello, World! "
  - print(a.lower()) #hello, world!

- upper() method returns the string in upper case:
  - a = "Hello, World!“
  - print(a.upper())#HELLO, WORLD!
- The replace() method replaces a string with another string:
  - print(a.replace("He","X"))#Xllo, World!
- The split() method splits the string into substrings if it finds instances of the separator:
  - a = "Hello, World!“
  - print(a.split(",")) # returns ['Hello', ' World!']
- Presence of a substring using in
  - if "llo" in a:
  -     print("Yes")
  - else:
  -     print("No")
  - Output: Yes

- String Concatenation: Merge variable a with variable b into variable c
  - a = "Hello"
  - b = "World“
  - c = a + b
  - print(c) #HelloWorld
  - For adding space in between
  - c=a+ “ ” +b #Hello World

See colab

Exercise

- Assume a, b and c are variables of type string
- a="Hello"
- b="World"
- c="This is my first program in string"

- Write a python code to merge these 3 strings to a single string so that value corresponding to each string variables appear in separate line. Print that merged string.
- Count total number of characters in merged string.
- Convert the merged string to lower case and print.
- Take the lower cased string and separate out all the words and print them.

```
a="Hello"
b="World"
c="This is my first program in string"
d=a+"\n"+b+"\n"+c
print(d)
print(len(d))
e=d.lower()
print(e)
print(e.replace("\n"," ").split(" "))
```

Output:

```
Hello
World
This is my first program in string
46
hello
world
this is my first program in string
['hello', 'world', 'this', 'is', 'my', 'first', 'program', 'in', 'string']
```

See colab

Sequences and collections

Datatype: tuple

- **tuple:** An ordered collection of n values of any type (n >= 0). It is **unchangeable**. In Python tuples are written with round brackets.
  - thistuple = ("a", "b", "c")
    Printing tuple: print(thistuple) #Output: ('a', 'b', 'c')
  - Access Tuple Items: print(thistuple[0]) #Output: a
  - It is unchangeable: a[0]=4 (error, not mutable)
  - Accessing a Range of Indexes, Negative Indexing are similar to string operation
  - Example:

1. thistuple = ("apple", "banana", "cherry")

   print(thistuple[-1])

   Output: cherry

2. thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
   print(thistuple[2:5])
   Output: ('cherry', 'orange', 'kiwi')
3. thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
   print(thistuple[-4:-1])
   Output: ('orange', 'kiwi', 'melon')

Check if Item Exists:

4.
```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
    print("Yes, 'apple' is in the fruits tuple")
```
Output: Yes, 'apple' is in the fruits tuple

Check length of the tuple:

5.
```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```
Output: 3

Join Two Tuples:

6.
```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)
tuple3 = tuple1 + tuple2
print(tuple3)
```
Output: ('a', 'b', 'c', 1, 2, 3)

See colab

Exercise

- Declare a tuple having following programming languages as items in order "C", "Python", "Java", "Pascal", "Cobol", "Fortran", "C++", "Perl". Find how many items are there in tuple. Check if "php " is there in the tuple.
- You are asked to store ip and port of your office computer in a tuple. Lets say ip and port is written using following convention 172.16.1.6:8080. Now you are asked to find out the port number from tuple.

```
tuple1 = ("C", "Python", "Java", "Pascal", "Cobol", "Fortran", "C++", "Perl")
print(len(tuple1))
if "php" in tuple1:
  print("Yes")
else:
  print("No")
```

Output:

```
8
No
```

See colab

```
iptuple=("172.16.1.6:8080",)
x=iptuple[0]
y=x.split(":")
print(y[1])
Output: 8080
```

See colab

Datatype: set

- **set:** A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.
  - a = {1, 2, 'a'}
  - Access Items: you can not access of a set referring the index as set is un-indexed. However, you may check membership using `in’ operator or you can use loop for printing all items though there is no guarantee in which order items will be printed.
  - Items of set are unique
  - a={1,1,2}
  - print(len(a))
  - Output: 2

See colab

Operations on sets

- *Intersection*
  - {1, 2, 3, 4, 5}.intersection({3, 4, 5, 6}) *# {3, 4, 5}*
  - {1, 2, 3, 4, 5} & {3, 4, 5, 6} *# {3, 4, 5}*
- *Union*
  - {1, 2, 3, 4, 5}.union({3, 4, 5, 6}) *# {1, 2, 3, 4, 5, 6}*
  - {1, 2, 3, 4, 5} | {3, 4, 5, 6} *# {1, 2, 3, 4, 5, 6}*
- *Difference*
  - {1, 2, 3, 4}.difference({2, 3, 5}) *# {1, 4}*
  - {1, 2, 3, 4} - {2, 3, 5} *# {1, 4}*

Operations on sets

- *Symmetric difference with (union – intersection)*
  - {1, 2, 3, 4}.symmetric_difference({2, 3, 5}) # *{1, 4, 5}*
  - {1, 2, 3, 4} ^ {2, 3, 5} # *{1, 4, 5}*
- *Superset check*
  - {1, 2}.issuperset({1, 2, 3}) # *False*
  - {1, 2} >= {1, 2, 3} # *False*
- *Subset check*
  - {1, 2}.issubset({1, 2, 3}) # *True*
  - {1, 2} <= {1, 2, 3} # *True*
- *Disjoint check*
  - {1, 2}.isdisjoint({3, 4}) # *True*
  - {1, 2}.isdisjoint({1, 4}) # *False*

Operations on sets

- *Existence check*
  - 2 **in** {1,2,3} *# True*
  - 4 **in** {1,2,3} *# False*
  - 4 **not in** {1,2,3} *# True*
- *Add and Remove*
  - s = {1,2,3}
  - s.add(4) *# s == {1,2,3,4}*
  - s.discard(3) *# s == {1,2,4}*
  - s.discard(5) *# s == {1,2,4} [discarding some element which is not a member does not generate any error]*
  - s.remove(2) *# s == {1,4} [however, removing some key generates error message if key does not exist]*
  - s.remove(2) *# KeyError!*

See colab

Exercise

- Given below 5 sets

Odds = {"1","3","5","7","9"}

Evens = {"2","4","6","8","10"}

Primes = {"3","5","7"}

Numbers = {"1","2","3","4","5","6","7","8","9","10"}

Squares = {"1","4","9"}

You are asked to find out following

I. set of numbers which are even and at the same time square.
II. Set of numbers which are either square or prime
III. Set of numbers which are odd but not primes
IV. Verify if set Odds is a superset of set Primes
V. Verify if set Evens is disjoint from set Primes

```
Odds = {"1","3","5","7","9"}
Evens = {"2","4","6","8","10"}
Primes = {"3","5","7"}
Numbers = {"1","2","3","4","5","6","7","8","9","10"}
Squares = {"1","4","9"}
inter_even_square = Evens & Squares
print(inter_even_square) # Output: {'4'}
square_or_prime = Squares | Primes
print(square_or_prime) # Output: {'3', '1', '5', '4', '7', '9'}
odd_but_not_prime = Odds - Primes
print(odd_but_not_prime) # Output: {'1', '9'}
if Odds >= Primes:
  print("Set Odds is superset of set Primes")
# Output: Set Odds is superset of set Primes
if Evens.isdisjoint(Primes):
  print("Set Evens and Primes are disjoint sets")
#Output: Set Evens and Primes are disjoint sets
```

See colab

Datatype:dict

- A dictionary in Python is a collection of key-value pairs. The dictionary is surrounded by curly braces. Each pair is separated by a comma and the key and value are separated by a colon. Here is an example

```
thisdict = {
 "brand": "Ford",
 "model": "Mustang",
 "year": 1964
}
```

Operations on Dictionary

1. **Printing a dictionary**:

   print(dictionary_name)

   Example:

   print(thisdict)

   Output: {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}

2. **Accessing Items**

   You can access the items of a dictionary by referring to its key name, inside square brackets

   x = thisdict["model"]

   print(x)

   Output: Mustang

Continue

3. **Change Values**

```
thisdict["year"] = 2018
```

4. **Check if key exist**

```
if "model" in thisdict:
    print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

5. **Finding dictionary length**

```
print(len(thisdict))
```

6. **Adding Items**

```
thisdict["color"] = "red"
print(thisdict)
```

Output: {'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}

Continue

7. **Removing Items**

```
thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}
```

thisdict.pop("model")

```
print(thisdict)
```

Output: {'brand': 'Ford', 'year': 1964}

8. **Removing entire dictionary**

```
del thisdict
```

See colab

Exercise

Assume a dictionary is defined as follows

```
mycar = {
  "brand": "Ford",
  "model": "Freestyle",
  "year": 2018
}
```

Add two more keys ("color" and "Enginetype") and corresponding values("white" and " Turbo Diesel") in this dictionary. Print modified dictionary. Check if there is any key "year " in mycar dictionary. If it is there then verify if the model is latest or not. Model will be said latest if it is a car of year 2021.

```
mycar = {
  "brand": "Ford",
  "model": "Freestyle",
  "year": 2018
}
mycar["color"]="white"
mycar["EngineType"]= " Turbo Diesel"
print(mycar)
#Output:{'brand': 'Ford', 'model': 'Freestyle', 'year': 2018, 'color': 'white',
'EngineType': 'Turbo Diesel'}
if "year" in mycar:
  print(mycar["year"]) #Output: 2018
  x=mycar["year"]
  if x == 2021:
    print("Latest Model")
  else:
    print("Old Model")
#Output: Old Model
```

See colab

Datatype:list

- list is more like an array in other languages
- int_list = [1, 2, 3]
- string_list = ['abc', 'defghi']
- A list can be empty:
- empty_list = []
- The elements of a list are not restricted to a single data type
- mixed_list = [1, 'abc', True, 2.34, None]
- A list can contain another list as its element:
- nested_list= [['a', 'b', 'c'], [1, 2, 3]]

Continue..

- The elements of a list can be accessed via an *index*
  - names = ['Alice', 'Bob', 'Craig', 'Diana', 'Eric']
  - print(names[0]) # Alice
  - print(names[2]) # Craig
- Indices can also be negative which means counting from the end of the list (-1 being the index of the last element).
  - print(names[-1]) # Eric
  - print(names[-4]) # Bob

Continue..

- Lists are mutable, so you can change the values in a list:
  - names[0] = 'Ann'
  - **print**(names)
  - *# Outputs ['Ann', 'Bob', 'Craig', 'Diana', 'Eric']*
- Besides, it is possible to add and/or remove elements from a list. Append object to end of list with L.append(object), returns None.
  - names = ['Alice', 'Bob', 'Craig', 'Diana', 'Eric']
  - names.append("Sia")
  - **print**(names)
  - *# Outputs ['Alice', 'Bob', 'Craig', 'Diana', 'Eric', 'Sia']*

Continue..

- Add a new element to list at a specific index. L.insert(index, object)
  - names.insert(1, "Nikki")
  - **print**(names)
  - *# Outputs ['Alice', 'Nikki', 'Bob', 'Craig', 'Diana', 'Eric', 'Sia']*
- Remove the first occurrence of a value with L.remove(value), returns None
  - names.remove("Bob")
  - **print**(names) *# Outputs ['Alice', 'Nikki', 'Craig', 'Diana', 'Eric', 'Sia']*

Continue..

- Get the index in the list of the first item whose value is x. It will show an error if there is no such item. *['Alice', 'Nikki', 'Craig', 'Diana', 'Eric', 'Sia']*
  - name.index("Alice")
  - 0
- Remove and return item at index (defaults to the last item) with L.pop([index]), returns the item
  - names.pop() *# Outputs 'Sia'*
- Count length of list
  - len(names)
  - 5

- The del keyword removes the specified index
  - thislist = ["apple", "banana", "cherry"]
    del thislist[0]
  - The del keyword can also delete the list completely
  - thislist = ["apple", "banana", "cherry"]
    del thislist
- The clear() method empties the list
  - thislist = ["apple", "banana", "cherry"]
    thislist.clear()

Continue..

- count occurrence of any item in list
  - a = [1, 1, 1, 2, 3, 4]
  - a.count(1)
  - 3
- Reverse the list
  - a.reverse()
  - [4, 3, 2, 1, 1, 1]

- Sort a list:
  - x=['m','n','a','c','b']
  - x.sort()
  - print(x)# ['a', 'b', 'c', 'm', 'n']

Check if Item Exists

- Check if "apple" is present in the list:
- thislist = ["apple", "banana", "cherry"]
  if "apple" in thislist:
      print("Yes, 'apple' is in the fruits list")

See colab

Exercise

- Assume that a list of languages are there in list names subjects as follows
- subjects = ["Python","C","Java","Basic","Cobol","Python","C++"]
  - Find the number of subjects in the subjects list
  - Print number of subjects in current list
  - Add new subject Perl at the end of the list
  - verify if the subject Perl added or not
  - Add a new subject "Pascal" at the sixth position of existing list
  - verify if the subject "Pascal" added or not at the sixth position
  - Verify if "Java" is there in the list
  - If "Java" is there then remove Java from the list
  - Print subjects in current list
  - remove last item of the list
  - Verify if the last item is removed from list or not
  - Print number of times "Python" appears in the subjects list

Exercise

```
1.  subjects = ["Python","C","Java","Basic","Cobol","Python","C++"]
2.  number_of_subjects = len(subjects)
3.  print("Line 3:",number_of_subjects)
4.  subjects.append("Perl")
5.  print("Line 5:",subjects)
6.  subjects.insert(5,"Pascal")
7.  print("Line 7:",subjects)
8.  if "Java" in subjects:
9.          subjects.remove("Java")
10.         print("Line 10:",subjects)
11. subjects.pop()
12. print("Line 12:",subjects)
13. print("Line 13:",subjects.count("Python"))
```

Output:

- Line 3: 7
- Line 5: ['Python', 'C', 'Java', 'Basic', 'Cobol', 'Python', 'C++', 'Perl']
- Line 7: ['Python', 'C', 'Java', 'Basic', 'Cobol', 'Pascal', 'Python', 'C++', 'Perl']
- Line 10: ['Python', 'C', 'Basic', 'Cobol', 'Pascal', 'Python', 'C++', 'Perl']
- Line 12: ['Python', 'C', 'Basic', 'Cobol', 'Pascal', 'Python', 'C++']
- Line 13: 2

See colab

Testing the type of variables

```
a = '123'
print(type(a))
# Out: <class 'str'>
b = 123
print(type(b))
# Out: <class 'int'>
```

```
l1=[1,2,3]
print(type(l1)) #output:<class 'list'>
s1={1,2,3}
print(type(s1)) #output:<class 'set'>
t1=(1,2,3)
print(type(t1))  #output:<class 'tuple'>
d1={"1":1,"2":2,"3":3}
print(type(d1)) #output:<class 'dict'>
```

See colab

Converting datatypes

- For example, '123' is of str type and it can be converted to integer using int function
- a = '123'
- b = int(a)
- Converting from a float string such as '123.456' can be done using float function.
- a = '123.456'
- b = float(a)
- c = int(a) *# ValueError*
- d = int(b) *# 123*

- You can also convert sequence or collection types
- a = 'hello'
- list(a) *# ['h', 'e', 'l', 'l', 'o']*
- set(a) *# {'o', 'e', 'l', 'h'}*
- tuple(a) *# ('h', 'e', 'l', 'l', 'o')*

See colab

Exercise

- Assume that a list of languages are there in list names subjects as follows
- subjects = ["Python","C","Java","Basic","Cobol","Python","C++"]
  - Print the list of unique subjects in subjects list
  - Find the number of unique subjects in subjects list

```
subjects = ["Python","C","Java","Basic","Cobol","Python","C++","Cobol"]
unique_subjects=set(subjects)
print(list(unique_subjects))
print(len(unique_subjects))
```

See colab

Python Operators

- Please go through on your own with list of Arithmetic Operators, Assignment Operators, Comparison Operators, Logical Operators, Identity Operators, Membership Operators
- https://www.w3schools.com/python/python_operators.asp

Conditional Statement

- Like other language python supports if, elif (else if) and else.
- Structure of if else looks like

```
if condition:
   body
else:
   body
```

- Examples:

You are supposed to check value of n and if it is greater than 2 then print "Not Small" and else print "Small"

```
n=5
if n>2:
   print("Not Small")
else:
   print("Small")
```

- Output: Not Small

- Examples:

    You are supposed to check value of n and if it is less than or equal to 2 then print "Small" if it is 3 or 4 print "Medium" else print "Big".

```
n=5
if n<2:
    print("Small")
elif n<4:
    print("Medium")
else:
    print("Big")
```

- Output:Big

Nested If else

We can use nested if else for doing the last example problem in following manner.

```
if n>2:
    if n>4:
        print("Big")
    else:
        print("Medium")
else:
    print("Small")
```

- Output: Big

See colab

Ternary Operator

Like C, Python also supports ternary operator. You can print a specific message when certain condition is evaluated to be true and other message if condition failed. Structure of ternary operator is like

message1 if condition else message2

When condition is satisfied message1 is printed else message2 is printed.

Example:

n = 5

"Greater than 2" **if** n > 2 **else** "Smaller than or equal to 2"

Output: 'Greater than 2‘

You can use nested ternary operation as well.

Example:

n = 5

"Hello" **if** n > 10 **else** "Goodbye" **if** n > 5 **else** "Good day "

Output: Good day

Membership Checking

- If item/key in list/tuple/set/dictionary:
  - Do whatever action you would like to perform

Loop

Python Loops

- Python has two primitive loop commands:
  - for loops: A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
  - while loops: With the while loop we can execute a set of statements as long as a condition is true.

Looping through string characters

- Example:

```
x="abcd"
for char in x:
        print(char)
```

Output:

```
a
b
c
d
```

Looping through tuple

- Example:

```
x=(1,2,3,4)
for item in x:
        print(item)
```

Output:

```
1
2
3
4
```

Looping Through a List

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
        print(x)
```

- Output

```
apple
banana
cherry
```

Looping Through a Set

- Example:

```
x={1,2,3,4}
for item in x:
        print(item)
```

Output:

```
1
2
3
4
```

Looping through dictionary

- Example:

```
x={1:"One",2:"Two",3:"Three",4:"Four"}
for item in x:
        print(item,":",x[item])
```

Output:

```
1 : One
2 : Two
3 : Three
4 : Four
```

Break statement in for loop

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
    if x == "banana":
        break
```

Output:

```
apple
banana
```

Continue statement with for loop

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    if x == "banana":
        continue
    print(x)
```

Output:

```
apple
cherry
```

The range() Function

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends before the specified number.

```
for x in range(6):
    print(x)
```

Output:

```
0
1
2
3
4
5
```

Specify starting value other than 0

The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):

```
for x in range(2,6):
    print(x)
```

Output:

```
2
3
4
5
```

Specify other than default increment value 1

The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 20, **3**):

```
for x in range(2, 20, 3):
    print(x)
```

Output:

```
2
5
8
11
14
17
```

Else in For Loop

- The else keyword in a for loop specifies a block of code to be executed when the loop is finished

```
for x in range(4):
  print(x)
else:
  print("Finally finished!")
```

Output:

```
0
1
2
3
Finally finished!
```

While Loops

Example: Print number 1 to 5.

```
i = 1
while i < 6:
    print(i)
    i += 1
```

Output:

```
1
2
3
4
5
```

Break statement with While

Example: Exit the loop when i is 3:

```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```

Output:

```
1
2
3
```

Continue statement with while loop

Example: Continue to the next iteration if i is 3:

```
i = 0
while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)
```

Output:

```
1
2
4
5
6
```

else statement

- The else Statement: With the else statement we can run a block of code once when the condition no longer is true
- Print a message once the condition is false:

```
i = 1
while i < 3:
   print(i)
   i += 1
else:
   print("i is no longer less than 3")
```

Output:

```
1
2
i is no longer less than 3
```

Nested Loop

- A nested loop is a loop inside a loop.
- The "inner loop" will be executed one time for each iteration of the "outer loop"

```
size = ["small","large"]
fruits = ["apple","banana","cherry"]
for s in size:
    for f in fruits:
        print(s,f)
```

Output:

```
small apple
small banana
small cherry
large apple
large banana
large cherry
```

Exercise

- A list of subjects taken by student (["Python","C","Java","Basic","Cobol","Python","C++","Cobol"]) is provided also a dictionary ({"Python":3,"C":3,"Java":4,"Basic":2,"Cobol":3,"C++":4, "Perl":3}) is provided which contains credit corresponding to each subject.
  - Check if there is any duplicate entry in subjects taken list.
  - Print unique subject list
  - Find out the total credits taken by the student.

Exercise

```
subjects_taken = ["Python","C","Java","Basic","Cobol","Python","C++","Cobol"]
subject_credit_dict = {"Python":3,"C":3,"Java":4,"Basic":2,"Cobol":3,"C++":4, "Perl":3}
subject_set=set(subjects_taken)
if len(subjects_taken) != len(subject_set):
    print("There are duplicate subjects in subject taken list")
    print("Subjects taken by the student are",subject_set)
total_credit=0
for subject in subject_set:
    total_credit+=subject_credit_dict[subject]
print("Total credit taken by the student is",total_credit)
```

- A list of subjects taken by student (["Python","C","Java","Basic","Cobol","C++"]) is provided also a dictionary ({"Python":3,"C":3,"Basic":2,"Cobol":3,"C++":4, " Perl":3, "Python":4}) is provided which contains credit corresponding to each subject. However, dictionary may have some duplicate entry, or might have some missing entry. See what happened when there are duplicate entries.
  – Check if there is any missing entry in dictionary, if it is there then add an entry for that with credit 4

- Output:
- There are duplicate subjects in subject taken list Subjects taken by the student are {'Basic', 'C++', 'Java', 'Python', 'Cobol', 'C'}
- Total credit taken by the student is 19

```
subjects_taken = ["Python","C","Java","Basic","Cobol","C++"]
subject_credit_dict = {"Python":3,"C":3,"Basic":2,"Cobol":3,"C
++":4, "Perl":3, "Fortran":3,"Python":4}
print(subject_credit_dict)  #Note that credit of Python is show
ing 4
subject_set=set(subjects_taken)

for subject in subject_set:
        if subject not in subject_credit_dict:
                subject_credit_dict[subject]=4
print(subject_credit_dict)
```

- {'Python': 4, 'C': 3, 'Basic': 2, 'Cobol': 3, 'C++': 4, 'Perl': 3, 'Fortran': 3}
- {'Python': 4, 'C': 3, 'Basic': 2, 'Cobol': 3, 'C++': 4, 'Perl': 3, 'Fortran': 3, 'Java': 4}

Functions

- A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

```
fruits = ["apple","banana","cherry"]
def add_fruit(name):
   fruits.append(name)
add_fruit("goava")
print(fruits)
Output:
['apple', 'banana', 'cherry', 'goava']
```

Default Parameter Value

```
def my_function(country = "India"):
    print("I am from " + country)

my_function("Sweden")
```

```
Output: I am from Sweden
```

```
my_function()
```

```
Output: I am from India
```

Return Values

```
def my_function(x):
    return 5 * x
print(my_function(3))
```

Output:

```
15
```

Exercise

- Write a function to check if a number is prime or not

Exercise

```
def prime_check(num):
    prime=1
    for i in range(2,num):
        if num%i == 0:
            print(num,"is not a prime number")
            prime = 0
            break
    if prime==1:
        print(num, "is a prime number")

prime_check(17)
```

See colab

User input

- username = input("Enter username:")
  print("Username is: " + username)
- Input is always in string format
- Let us redo the last exercise once again where input is taken from user

```
def prime_check(num):
  prime=0
  for i in range(2,num):
    if num%i == 0:
      print(num,"is not a prime number")
      prime = 1
      break
  if prime!=1:
    print(num, "is a prime number")

x=int(input("Enter number to be checked for primality"))
prime_check(x)
```

See colab

Module

- What is a Module?
  - Consider a module to be the same as a code library.
  - A file containing a set of functions, variables that you want to include in your application.

Create and Use a Module

- Save this code in a file named mymodule.py

```
def greeting(name):
    print("Hello, " + name)
```

- Import the module named mymodule, and call the greeting function:

```
import mymodule
mymodule.greeting("Sourav")
```

Output:

```
Hello, Sourav
```

- Or we may import a specific function from a module

```
from mymodule import greeting
greeting("Sourav")
Hello, Sourav
```

Variables in Module

- The module can contain functions, as already described, but also variables of all types (arrays, dictionaries, objects etc):
- Save this code in the file mymodule.py

```
person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

- Import the module named mymodule, and access the person1 dictionary:

```
import mymodule

a = mymodule.person1["age"]
print(a)
```

Output:

```
36
```

Exercise

- Improve the primality checking exercise by using math library sqrt function.

```
from math import sqrt
def prime_check(num):
  prime=1
  x=int(sqrt(num))
  for i in range(2,x+1):
    if num%i == 0:
      print(num,"is not a prime number")
      prime = 0
      break
  if prime==1:
    print(num, "is a prime number")

x=int(input("Enter number to be checked for primality"))
prime_check(x)
```

See Colab

Important Library for Data Science

- **Numpy**: N-dimensional arrays, Matrices and Linear Algebra
- **Scipy**: Algorithms from linear algebra, optimization, statistics and signal processing
- **Pandas**: Data Manipulation and Analysis
- **Matplotlib**: Data Visualization
- **IPython**: Interactive shell for Python
- **Scikit-learn**: Machine Learning

Python Class and Objects

- **Class**: Class creates a user-defined data structure, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

  - Attributes are the variables that belong to a class.

- **Class Objects:** An Object is an instance of a Class.
  - State: It is represented by the attributes of an object.
  - Behavior: It is represented by the methods of an object.
  - Identity: It gives a unique name to an object

Example

```
class Human:
        attr1 = "mammal"
        attr2 = “human"
        def fun(self):
                print("I'm a ", self.attr1)
                print("I'm a", self.attr2)
Alex = Human()

print(Alex.attr1)
Alex.fun()
```

Output:

```
mammal
I'm a mammal
I'm a human
```

- **The self**
  - Class methods must have an extra first parameter in the method definition.
  - If there is any method which does not take any argument, in that case also this (self) argument will be there

Python Classes and Objects

- **The __init__() Function:** The __init__ method is similar to constructors in C++ and Java. Constructors are used to initializing the object’s state. Like methods, a constructor also contains a collection of statements(i.e. instructions) that are executed at the time of Object creation. It runs as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object.

- Example:

  Create a class named Person, use the __init__() function to assign values for name and age:

```
class Person:
    def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)
print(p1.name)
print(p1.age)
```

Python Classes and Objects

- Methods: Class can also contain other methods. Methods in class are functions that can be accessed by objects of this class.

Example

Let us create a method in the Person class:

```
class Person:
	def __init__(self, name, age):
		self.name = name
		self.age = age

	def myfunc(self):
		print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

Output: Hello my name is John

Python Classes and Objects

- The self Parameter: The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.
- It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class:

Example

Use the words *mysillyobject* and *abc* instead of *self*:

```
class Person:
        def __init__(mysillyobject, name, age):
                mysillyobject.name = name
                mysillyobject.age = age

        def myfunc(abc):
                print("Hello my name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

Output: Hello my name is John

Python Classes and Objects

- Modify Object Properties: You can modify properties on objects like this:
  Example
  Set the age of p1 to 40:
  p1.age = 40

- Delete Object Properties: You can delete properties on objects by using the del keyword:
  Example
  Delete the age property from the p1 object:
  del p1.age

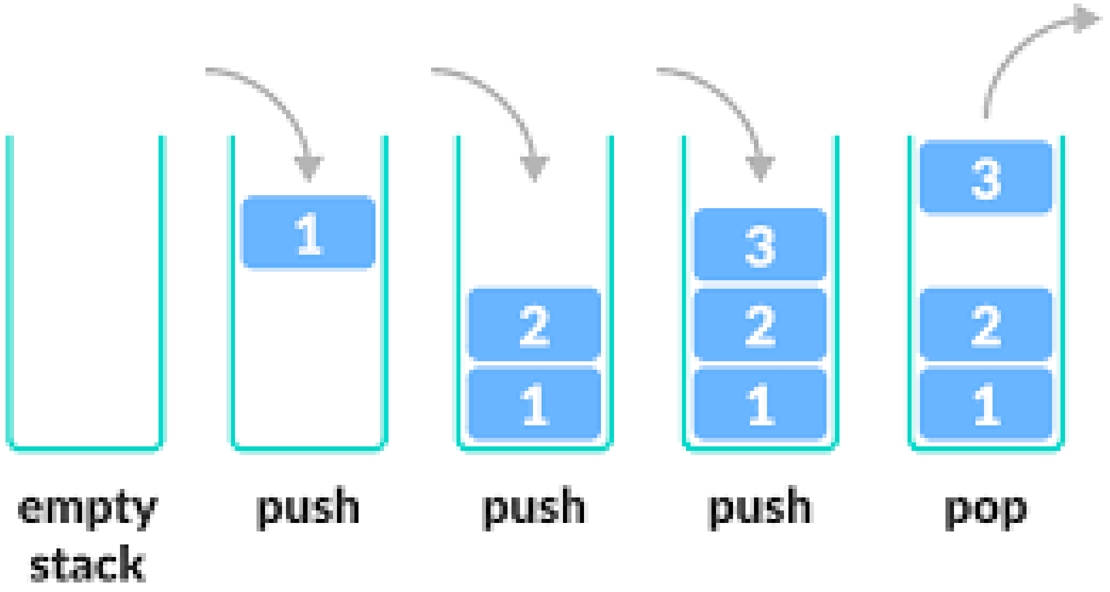
- Delete Objects: You can delete objects by using the del keyword:
  Example
  Delete the p1 object:
  del p1

Data Structure Using Python

- Stack, Queue, Linked List

Stack

- Stack: Last in Fast Out (LIFO)



Creating a library for Stack

```
class Stack:
        def __init__(self):
                self.items = []

        def isEmpty(self):
                return self.items == []

        def push(self, item):
                self.items.append(item)

        def pop(self):
                return self.items.pop()

        def top(self):
                return self.items[len(self.items)-1]

        def size(self):
                return len(self.items)
```

Example

- Use Stack library and create an empty stack then check if the stack is empty. Then push 1,2 and 3 into stack and print the content of stack. Pop one element and again check the content of stack.

```
From Stack import Stack
s=Stack()
if s.isEmpty():
        print("Stack is empty")
 s.push(1)
 s.push(2)
 s.push(3)
 s.print()
 s.pop()
 s.print()
```

See Colab

Exercise

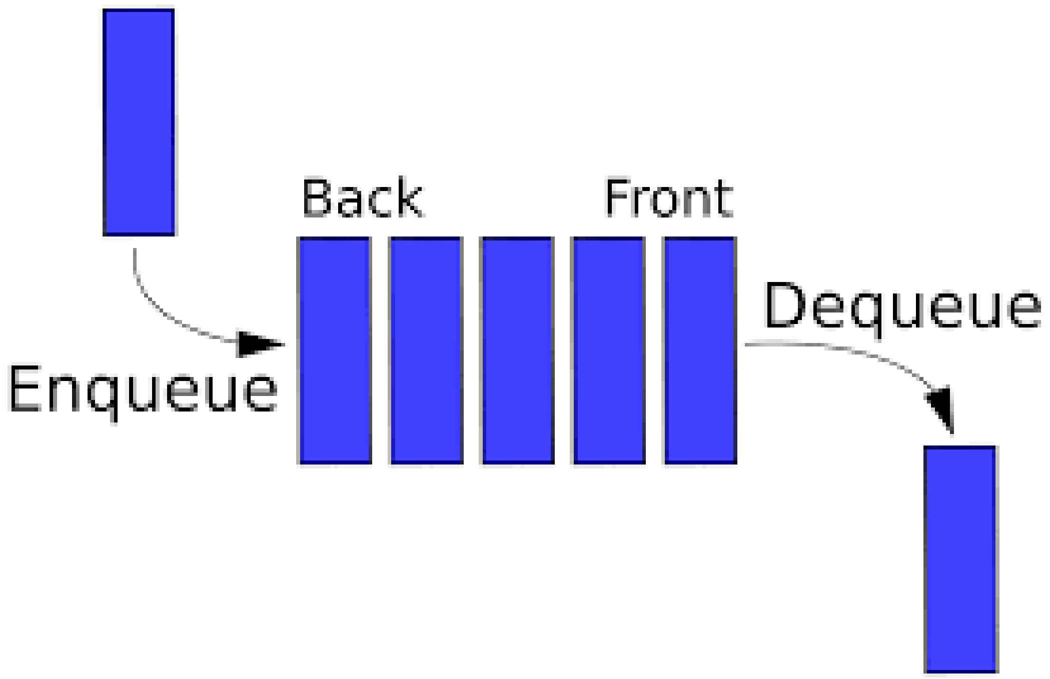
Use Stack library to create an empty stack. Take input from user that how many integers need to be inserted. And insert that many integers using input from user. Write a function which takes one stack as input parameter and return a reversed stack.

```
from Stack import Stack
def stack_reversal(s):
  s1=Stack()
  if s.size() == 0:
    return s
  else:
    while s.isEmpty() != 1:
      s1.push(s.pop())
    return s1

s=Stack()
x=int(input("Enter number of elements you want to push into stack"))
for i in range(x):
  item=int(input("Enter number to be pushed into stack"))
  s.push(item)
s.print()
s=stack_reversal(s)
s.print()
```

See Colab

Queue

- Queue: First in First Out (FIFO)



Queue Library

```
class Queue:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def enqueue(self, item):
        self.items.insert(0,item)

    def dequeue(self):
        return self.items.pop()

    def size(self):
        return len(self.items)

    def print(self):
        print(self.items)
```

Example

- Use Queue library and create an empty queue then check if the queue is empty. Then enqueue 1,2 and 3 into queue and print the content of queue. Dequeue one element and again check the content of queue.

```
from Queue import Queue
q=Queue()
if q.isEmpty() == 1:
      print("Queue is empty")
 q.enqueue(1)
 q.enqueue(2)
 q.enqueue(3)
 q.print()
 q.dequeue()
 q.print()
```

See Colab

Exercise

- Use Queue library to create an empty queue. Populate that queue. Now take an input parameter from user for existence/count checking in queue.
- Use a stack to reverse the content of queue.

```
from Queue import Queue
q=Queue()
q1=Queue()
q.enqueue(1)
q.enqueue(2)
q.enqueue(5)
q.enqueue(2)
i_str=input("Enter element for count checking")
i=int(i_str)
count=0
while q.isEmpty() !=1:
  x=q.dequeue()
  q1.enqueue(x)
  if x == i:
    count+=1
if count >= 1:
  print("Element",i, " is present in queue",count, "times")
else:
  print("Element is not there in queue")
q=q1
del q1
q.print()
```

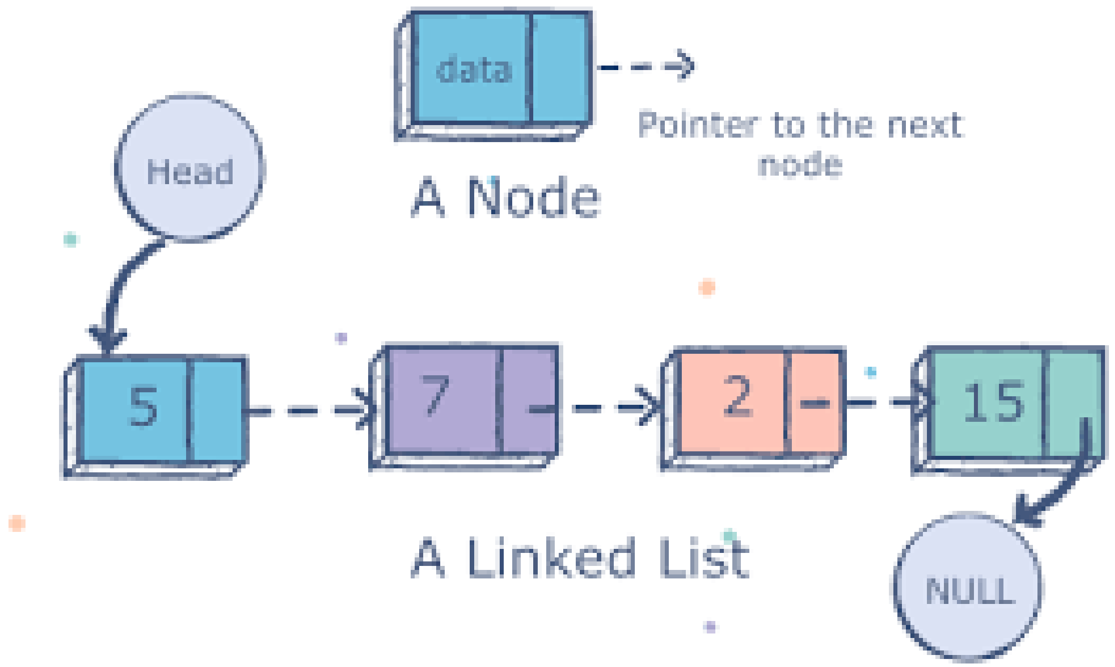
See Colab

```
from Stack import Stack as St
from Queue import Queue as Q
s1=St()
q1=Q()
x=int(input("Number of elements in Queue "))
for i in range(x):
        print("Enter element",i+1)
        item=int(input(""))
        q1.enqueue(item)
q1.print()
while q1.isEmpty() != True:
        s1.push(q1.dequeue())
while s1.isEmpty() != True:
        q1.enqueue(s1.pop())
del s1
q1.print()
```

See Colab

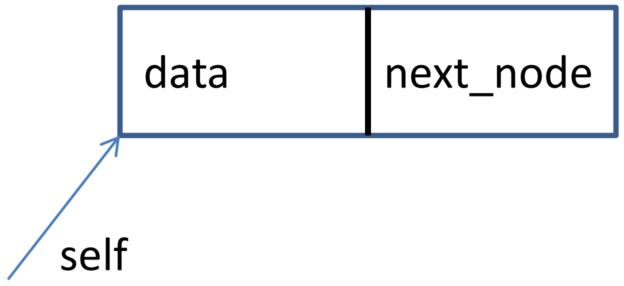
Linked List

- Linked List: A number of records which are linked in a sequential manner

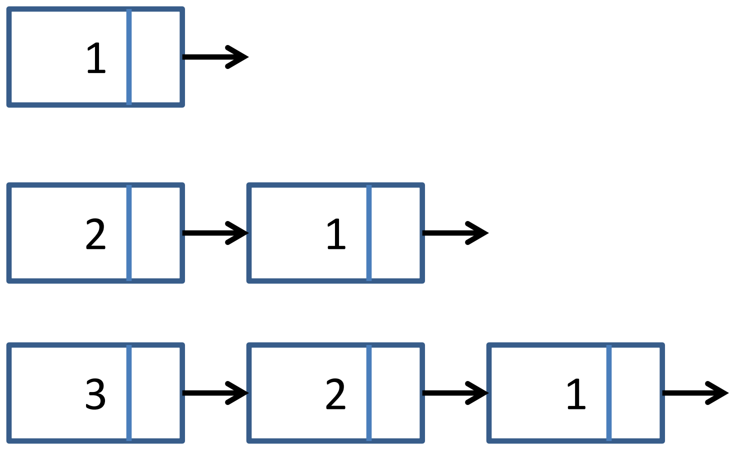


Library for Linked List

```
class Node:
    def __init__(self, data=None, next_node=None):
        self.data = data
        self.next_node = next_node

    def get_data(self):
        return self.data

    def get_next(self):
        return self.next_node

    def set_next(self, new_next):
        self.next_node = new_next
```

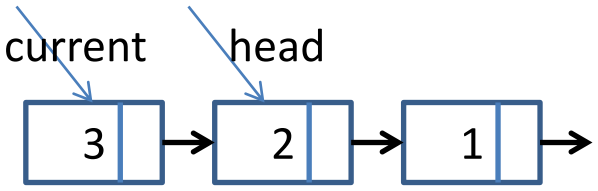


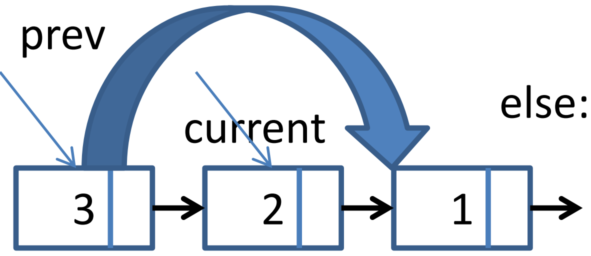
```
class LinkedList:
    def __init__(self, head=None):
        self.head = head

    def insert(self, data):
        new_node = Node(data)
        new_node.set_next(self.head)
        self.head = new_node

    def size(self):
        current = self.head
        count = 0
        while current:
            count += 1
            current = current.get_next()
        return count
```



```
def search(self, data):
    current = self.head
    found = False
    while current and found is False:
        if current.get_data() == data:
                found = True
                print("Found")
        else:
                current = current.get_next()
    if current is None:
        print("Data not in list")
    return current
```

```
def delete(self, data):
        current = self.head
        if current is None:
                print("Empty List")
                return
        previous = None
        found = False
        while current and found is False:
                if current.get_data() == data:
                        found = True
                else:
                        previous = current
                        current = current.get_next()
        if current is None:
                print("Data not in list")
                return
        if previous is None:
                self.head = current.get_next()
                del current
        else:
                previous.set_next(current.get_next())
                del current
```





```
 def delete_head(self):
         current = self.head
         if current != None:
                 self.head=current.get_next()
         return current

def print(self):
         current=self.head
         while current:
                 print(current.get_data())
                 current = current.get_next()
```

Example

- Import linkedlist class from linkedlist library. Use it for creating a linked list with elements 3,5,2,1. Find the size of the list and search for element

```
from LinkedList import LinkedList as LL
x=LL()
x.insert(1)
x.insert(2)
x.insert(5)
x.insert(3)
print("Size of current list is ",x.size())
print("Content of current list is")
x.print()
x.search(3)
x.delete(2)
x.print()
```

```
Output:
Size of current list is 4
Content of current list is
3
5
2
1
Found
3
5
1
```

See Colab

Exercise

- Create an empty list using LinkedList library. Write a code which will ask inputs from user to be inserted in the list. User can insert as many elements as he likes. Print the list. Reverse the list and print it again.

```
from LinkedList import LinkedList as LL
l1=LL()
l2=LL()
while 1:
        x=int(input("Enter element to be inserted "))
        l1.insert(x)
        y=int(input("Want to insert more then press 1 else 0 "))
        if y == 0:
                break
l1.print()
while 1:
        n = l1.delete_head()
        if n != None:
                l2.insert(n.get_data())
        else:
                break
l2.print()
```

Question??

Thank You!!