Lecture 13-14

Introduction to python

Define a variable

- A **variable** is a named storage location that can store a value of a particular data type.
- In other words, a variable has a name, a type and stores a value.
- In order to use a variable in python we don't need to declare and specify its data type.
- The syntax to defining a new variable is to write the name of the variable and pass it a value
  - nameOfTheVariable=value

Define a variable

- Examples:
  - Number=3
  - floatNumber=3.14
  - String =“hello”

```
In [3]: Number=3
        floatNumber=3.14
        String ="hello"
```

Define a variable

- Boolean:
  - X=True

Name of the variable

An identifier is needed to name a variable python imposes the following rules on identifiers:

- An identifier is a sequence of characters, comprising uppercase and lowercase letters (a-z, A-Z), digits (0-9), and underscore "_".
- White space (blank, tab, new-line) and other special characters (such as +, -, *, /, @, &, commas, etc.) are not allowed.

Name of the variable

- White space (blank, tab, new-line) and other special characters (such as +, -, *, /, @, &, commas, etc.) are not allowed.
- Examples

```
]: float-Number=3.14
```

```
  File "<ipython-input-8-40c038b465e0>", line 1
    float-Number=3.14
    ^
SyntaxError: cannot assign to operator
```

```
1Number =3
```

```
  File "<ipython-input-6-cb49eed17bb6>", line 1
    1Number =3
     ^
SyntaxError: invalid syntax
```

```
In [9]: @Number =3
```

```
  File "<ipython-input-9-797b639c16e8>", line 1
    @Number =3
            ^
SyntaxError: invalid syntax
```

Declaration of variables

- python imposes the following rules on identifiers:
  - An identifier must begin with a letter or underscore. It cannot begin with a digit.
  - Identifiers are case-sensitive. A rose is NOT a Rose, and is NOT a ROSE.

Print data on the screen

- print is used to print data on the screen.
- Syntax:
  - print()
- Example 1:

```
In [1]: print("hello world")
        hello world
```

- Example 2:

```
intvalue=3
floatNumber=3.14
String ="hello"
print(intvalue,floatNumber,String)
```

```
3 3.14 hello
```

Take input

- input is used to take the input from user.
- syntax
  - input()
- Example:

```
value = input("enter a number ")
```

```
enter a number 2
```

Check data type

- To determine a variable's type in Python you can use the type() function. The value of some objects can be changed. Objects whose value can be changed are called mutable and objects whose value is unchangeable (once they are created) are called immutable. Here are the details of Python datatypes

- Data type:

```
type(value)
```

```
str
```

Change data type of the input

- for integer value use the built-in-function
  - int()
- Example:

```
value = int(input("enter a number "))
```

```
enter a number 2
```

```
type(value)
```

```
int
```

Change data type of the input

- for integer value use the built-in-function
  - float()
- Example:

```
: value = float(input("enter a number "))
  enter a number 2

: type(value)
: float
```

Define a variable

- Boolean:

```
x=True
print(type(x))
```

- Examples:

```
<class 'bool'>
```

Boolean operator

- +, -, *, /,**

```
a=5
b=4
print(a+b)
print(a-b)
print(a**b)
```

```
9
1
625
```

Python Reserve words:

- The following identifiers are used as reserved words of the language, and cannot be used as ordinary identifiers.

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	el	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Special characters in strings

- The backslash (\) character is used to introduce a special character. See the following table.

Escape sequence	Meaning
\n	Newline
\t	Horizontal Tab
\\	Backslash

- \n:

```
print("What is your \n Name")
```

- Output

```
What is your
 Name
```

Special characters in strings

- The backslash (\) character is used to introduce a special character. See the following table.

Escape sequence	Meaning
\n	Newline
\t	Horizontal Tab
\\	Backslash

- \t:

```
print("What is your \t Name")
```

- Output

```
What is your     Name
```

Special characters in strings

- The backslash (\) character is used to introduce a special character. See the following table.

Escape sequence	Meaning
\n	Newline
\t	Horizontal Tab
\\	Backslash

- \\:

```
print("What is your \\ Name")
```

- Output

```
What is your \ Name
```

Conditional statements

- if-else
- Syntax:

```
if cond:
    # body
else:
    # else body
```

- Example:

```
if 5>3:
    print("5 is greater" )
else:
    print("3 is greater")
```

```
5 is greater
```

Indentation

- Python uses whitespace (spaces and tabs) to define program blocks whereas other languages like C, C++ use braces ({}) to indicate blocks of codes for class, functions or flow control.
- The number of whitespaces (spaces and tabs) in the indentation is not fixed, but all statements within the block must be the indented same amount.

Indentation

- Example:

```
a=5.0
b=4.5
print(a)
print(type(a))
if a>b:
    print("a>b")
  print("hello")
```

- Error:

```
    print("hello")
                ^
IndentationError: unindent does not match any outer indentation level
```

Conditional statements

- if-elif-else
- Syntax:

```
if cond:
    # body
elif cond:
    # else if body
else:
    # else body
```

- Example:

```
i=4
j=4
if i>j:
    print(i," is greater" )
elif i<j:
    print(j," is greater" )

else:
    print(i," is equal to",j)
```

```
4  is equal to 4
```

Repetitive structures

- while and for loop
- Syntax:

```
In [ ]: while cond:
            #body
```

- Example:

```
i=10
j=4
while i>j:
    print("value of j is",j)
    j+=1
```

```
value of j is 4
value of j is 5
value of j is 6
value of j is 7
value of j is 8
value of j is 9
```

Repetitive structures

- for loop
- Syntax:

```
for cond:
    #body
```

- Example:

```
for i in range(9):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
```

Repetitive structures

- for loop
- Syntax:

```
for cond:
    #body
```

- Example:

```
for i in "khola":
    if i=='a':
        print("a is in string")
```

```
a is in string
```

Functions

- def
- Syntax:

```
def nameofthefun():
    #body
```

Example:

```
def sum(a,b):
    print(a+b)
sum(4,5)
```

```
9
```

Example:

```
In [59]: def fun():
             print('Hello')
         fun()
```

```
Hello
```

Functions

Default Parameter Value:

The following example shows how to use a default parameter value.

If we call the function without parameter, it uses the default value:

Example

```
In [60]: def fun(Name="Khola"):
             print("My name is ", Name)
         fun("Ali")
         fun("Kiran")
         fun()
```

```
My name is  Ali
My name is  Kiran
My name is  Khola
```

Functions

- Return a value

The following example :
Example

```
In [63]: def val(num):
             return num*6
         print(val(6))
```

```
36
```

Built in functions

- max():

```
max_number=max(1,2,3)
print(max_number)
```

```
3
```

- min()

```
min_number = min(1,2,3,-1)
print(min_number)
```

```
-1
```

- upper()

```
word="Khola"
print(word.upper())
```

```
KHOLA
```

Built in functions

- lower:

```
word="Khola"
print(word.lower())
```

```
khola
```

Built in functions

- input()
- type()
- int()
- float()
- range()
- Max()
- Min()
- Upper()
- Lower()

String indices and accessing string elements

- Strings are arrays of characters and elements of an array can be accessed using indexing.
- Indices start with 0 from left side and -1 when starting from right side.
- string1 ="PYTHON TUTORIAL“

Character	P	Y	T	H	O	N		T	U	T	O	R	I	A	L
Index (from left)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Index (from right)	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

- See the following statements to access single character from various positions.

String indices and accessing string elements

- string1 ="PYTHON TUTORIAL“

```
In [3]: string1 ="PYTHON TUTORIAL"
```

- Check: string1[0]

```
In [5]: print(string1[0])
P
```

- Check: string1[-1]

```
In [6]: print(string1[-1])
L
```

- Check:

```
In [7]: print(string1[-5])
O
```

```
In [8]: print(string1[5])
N
```

```
In [9]: print(string1[6])
```

String indices and accessing string elements

- string1 ="PYTHON TUTORIAL“

```
In [3]: string1 ="PYTHON TUTORIAL"
```

- Check: string1[0]

```
In [5]: print(string1[0])
P
```

- Check: string1[-1]

```
In [6]: print(string1[-1])
L
```

- Check:

```
In [7]: print(string1[-5])
O
```

```
In [8]: print(string1[5])
N
```

```
In [9]: print(string1[6])
```

String indices and accessing string elements

- string1 ="PYTHON TUTORIAL“
- Check: string1[15]

```
In [13]: print(string1[15])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[13], line 1
----> 1 print(string1[15])

IndexError: string index out of range
```

- string1[-16]

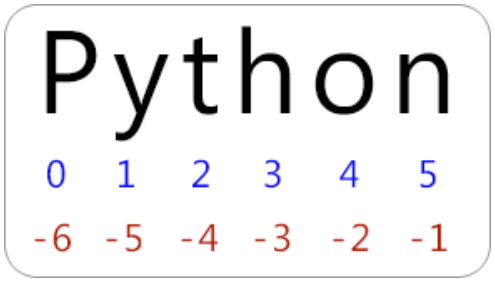
```
In [17]: print(string1[-16])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[17], line 1
----> 1 print(string1[-16])

IndexError: string index out of range
```

String Slicing

- To cut a substring from a string is called string slicing. Here two indices are used separated by a colon (:)
- A slice 3:7 means indices characters of 3rd, 4th, 5th and 6th positions. The second integer index i.e. 7 is not included.
- You can use negative indices for slicing. See the following statements.




```
In [18]: print(string1[1:4])
YTH
```

```
In [43]: print(string1[-4:-1])
RIA
```

List:

- list is a collection of elements, but each element is a scalar.
- A list is a container which holds comma-separated values (items or elements) between square brackets where Items or elements need not all have the same type.

```
In [19]: li=[5,6,7,8,9,1,2]

In [21]: print(li)
         [5, 6, 7, 8, 9, 1, 2]
```

- String list:

```
In [22]: li2=["Have","a ",'good',"day"]

In [23]: li2
Out[23]: ['Have', 'a ', 'good', 'day']
```

List:

- Items or elements need not all have the same type.

```
In [24]: Li3=["Khola",1,5.0,True]

In [25]: Li3
Out[25]: ['Khola', 1, 5.0, True]
```

- list

```
print(type(Li3[1]))
print(type(Li3))
print(type(Li3[2]))
```

```
<class 'int'>
<class 'list'>
<class 'float'>
```

List indices:

- List indices work the same way as string indices, list indices start at 0. If an index has a positive value it counts from the beginning and similarly it counts backward if the index has a negative value.
- As positive integers are used to index from the left end and negative integers are used to index from the right end, so every item of a list gives two alternatives indices. Let create a list called color_list with four items.
- color_list=["RED", "Blue", "Green", "Black"]

Item	RED	Blue	Green	Black
Index (from left)	0	1	2	3
Index (from right)	-4	-3	-2	-1

- If you give any index value which is out of range then interpreter creates an error message. See the following statements.

List indices:

- Items or elements need not all have the same type.

```
In [33]: Li3=["Khola",1,5.0,True,1,2,3]

In [34]: Li3
Out[34]: ['Khola', 1, 5.0, True, 1, 2, 3]
```

- List indices

```
In [37]: print(Li3[1:5])
         [1, 5.0, True, 1]
```

- Negative indices:

```
In [44]: print(Li3[-4:-1])
         [True, 1, 2]
```

List indices:

- Items or elements need not all have the same type.

```
In [33]: Li3=["Khola",1,5.0,True,1,2,3]

In [34]: Li3
Out[34]: ['Khola', 1, 5.0, True, 1, 2, 3]
```

- Index out of range

```
In [52]: print(Li3[8])
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[52], line 1
----> 1 print(Li3[8])

IndexError: list index out of range
```

List:

- Empty list:

```
In [53]: li=[]
```

- **list()** method

```
In [56]: li4=list(range(5))

In [57]: li4
Out[57]: [0, 1, 2, 3, 4]
```

- Find the length of the list: **len()**

```
In [1]: li4=list(range(5))

In [2]: li4
Out[2]: [0, 1, 2, 3, 4]

In [3]: print(len(li4))
        5
```

List:

- Empty list:

```
In [53]: li=[]
```

- list() method: Empty list

```
In [8]: li6=list()
```

- Change the range: even number list

```
li5=list(range(2,20,2))
```

```
li5
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18]
```

List:

- Iterate through the elements of the list:

```
In [11]: for i in li5:
             print(i)
```

```
2
4
6
8
10
12
14
16
18
```

- Iterating by index of sequences:

```
In [58]: for i in range(len(li5)):
             print(li5[i])
```

```
2
4
6
23
8
12
14
16
18
5
good
[0, 1, 2, 3, 4]
```

List:

- Iterate through the elements of the list:

```
In [11]: for i in li5:
             print(i)

2
4
6
8
10
12
14
16
18
```

- The length of a list may vary during execution. New elements may be added to the list, while others may be removed from it. This means that the list is a very dynamic entity.

```
In [12]: print(len(li5))
         del li5[4]
         print(li5)
         print(len(li5))

9
[2, 4, 6, 8, 12, 14, 16, 18]
8
```

List:

- Add a new element in a list:
  - Append →list.append(value)→insert at the end
  - insert→ list.insert(location, value)→insert at a specific location
- Append()

```
In [13]: print(li5)
         li5.append(5)
         print(li5)

         [2, 4, 6, 8, 12, 14, 16, 18]
         [2, 4, 6, 8, 12, 14, 16, 18, 5]
```

- Insert()

```
In [16]: li5.insert(3,23)

In [19]: li5

Out[19]: [2, 4, 6, 23, 8, 12, 14, 16, 18, 5]
```

List:

- Add a new element in a list:
  - Append →list.append(value)→insert at the end
  - insert→ list.insert(location, value)→insert at a specific location
- Append()

```
In [13]: print(li5)
         li5.append(5)
         print(li5)

         [2, 4, 6, 8, 12, 14, 16, 18]
         [2, 4, 6, 8, 12, 14, 16, 18, 5]
```

- Example:

```
In [20]: li5.append('good')
         print(li5)

         [2, 4, 6, 23, 8, 12, 14, 16, 18, 5, 'good']
```

List:

- Add a new element in a list:
  - Append →list.append(value)→insert at the end
  - insert→ list.insert(location, value)→insert at a specific location
- Append()

```
In [13]: print(li5)
         li5.append(5)
         print(li5)

         [2, 4, 6, 8, 12, 14, 16, 18]
         [2, 4, 6, 8, 12, 14, 16, 18, 5]
```

- Example:

```
In [23]: li5.append(li4)

In [24]: li5

Out[24]: [2, 4, 6, 23, 8, 12, 14, 16, 18, 5, 'good', [0, 1, 2, 3, 4]]
```

List:

- Check equality:
  - Create two lists and check if they are equal

List:

- Check equality:
  - Create two lists and check if they are equal

```
In [29]: li=[1,2,3]
         li2=[1,2,4]

In [30]: li==li2

Out[30]: False
```

- Check again:

```
In [31]: li=[1,2,3]
         li2=[1,2,3]

In [32]: li==li2

Out[32]: True
```

List: Some Bulit in functions

- Count the an element:

```
In [39]: li7=[1,2,3,4,1,2,5,6,7,8,9,8,1]

In [43]: print(li7.count(2))
2
```

- Sort():

```
In [44]: li7.sort()

In [45]: li7
Out[45]: [1, 1, 1, 2, 2, 3, 4, 5, 6, 7, 8, 8, 9]
```

List: Some Bulit in functions

- Count the an element:

```
In [39]: li7=[1,2,3,4,1,2,5,6,7,8,9,8,1]

In [43]: print(li7.count(2))
2
```

- Sort():

```
In [44]: li7.sort()

In [45]: li7
Out[45]: [1, 1, 1, 2, 2, 3, 4, 5, 6, 7, 8, 8, 9]
```

- In reverse order

```
In [47]: li7.sort(reverse=True)
         li7
Out[47]: [9, 8, 8, 7, 6, 5, 4, 3, 2, 2, 1, 1, 1]
```

List:

- Copy a list to another list:

```
In [48]: li8=[1,2,3]

In [49]: li7=li8

In [50]: print(li7)
         print(li8)

[1, 2, 3]
[1, 2, 3]
```

- Problem:

```
[1, 2, 3]

In [51]: li8.append(8)

In [52]: print(li7)
         print(li8)

[1, 2, 3, 8]
[1, 2, 3, 8]
```

List:

- Copy a list to another list: sol

```
In [53]: li7=li8[:]

In [54]: li7.append(21)

In [55]: print(li7)
         print(li8)

         [1, 2, 3, 8, 21]
         [1, 2, 3, 8]
```

Functions

- Passing a list as parameter:

  if you send a List as a parameter, it will still be a List when it reaches the function:
  Example

```
In [61]: def number(Li):
             for i in Li:
                 print(i)

         number([1,2,3,4,5])
```

```
1
2
3
4
5
```

Import:

- Add libraries, header files:
  - import
  - E.g:
    - import math

```
import math as mt
```

```
mt.sqrt(3)
```

```
1.7320508075688772
```

```
x = mt.ceil(1.4)
y = mt.floor(1.4)

print(x)
print(y)
```

```
2
1
```

Import:

- Add libraries, header files:
  - numpy
  - matplotlib
  - pandas
  - Sklearn

Task:

- Even number between 1 to 20
- Ask the user to enter a string(str) and a character(c) and find how many times the character(c) appears in the string?
- Write a function that will return all the vowels in a string
- Write a program that inputs salary. If salary is greater than or equal to 30000, then deducts 7% of salary. If salary is 20000 or more but less than 30000, then deduct 5% of salary. If salary is less than 20000 then deduct nothing. Print the net salary.