Practicum in human intelligent information processing

Vishal Gaurav

Python

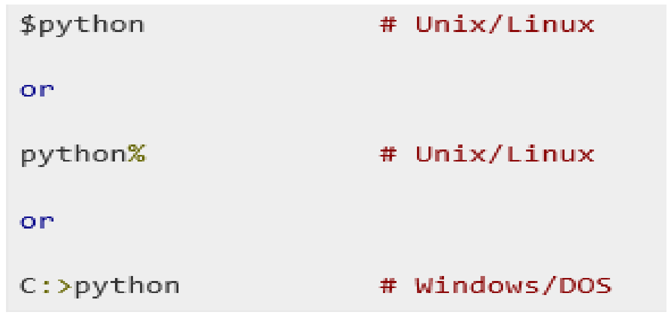
- **Interpreted**
- **Interactive**
- **Object-Oriented**

Table of content

Running Python

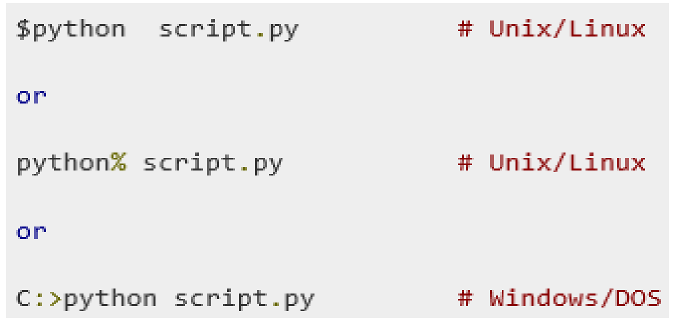
There are three different ways to start Python:

Interactive Interpreter: Enter **python** the command line.



```
$python                 # Unix/Linux
or
python%                 # Unix/Linux
or
C:>python               # Windows/DOS
```

Script from the Command-line: Python script can be executed at command line



```
$python  script.py           # Unix/Linux
or
python% script.py            # Unix/Linux
or
C:>python script.py          # Windows/DOS
```

Integrated Development Environment: Run python form a Graphical User Interface (GUI)

Execute Python scripts

Type the text below into a text editor and save as **hello.py.** Python files usually have the **.py** extension.

```
#!/usr/bin/env python
print('hello world')
```

To **start** the program, we have to open the command line and type: python hello.py

Variables

A variable is symbolic name for an area in memory that has a certain format. The format of that area is the area's type. It is called a variable because the value assigned to it can be changed. Due to the flexible nature of variable binding in Python, a variable can be rebound to any type at any point in a program.

Examples of valid variable names:

```
myVariable
Var1
X
x
_x
```

Examples of invalid variable names

1var – wrong, begins with digit
My#var- wrong

variable names can only include a-z, A-Z, _, and 0-9. Other special characters are not permitted.

Variables

Standard Data types.

- Numbers
- String
- List
- Tuples
- Dictionary

```
#!/usr/bin/python
counter = 100 # An integer assignment
miles = 1000.0 # A floating point
name = "John" # A string
print counter
print miles
print name
```

Multiple assignment

```
a=b=c=1
a, b, c = 1, 2, "john"
```

Strings and text

Try the program below:

```
#!/usr/bin/python
str = 'Hello World!'
print str # Prints complete string
print str[0] # Prints first character of the string
print str[2:5] # Prints characters starting from 3rd to 5th
print str[2:] # Prints string starting from 3rd character
print str * 2 # Prints string two times
print str + "TEST" # Prints concatenated string
```

Lists

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). The items belonging to a list can be of different data type.
The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

```
#!/usr/bin/python
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']
print list # Prints complete list
print list[0] # Prints first element of the list
print list[1:3] # Prints elements starting from 2nd till 3rd
print list[2:] # Prints elements starting from 3rd element
print tinylist * 2 # Prints list two times
print list + tinylist # Prints concatenated lists
```

Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

Differences between lists and tuples

Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as **read-only** lists.

```
#!/usr/bin/python
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
tinytuple = (123, 'john')
print tuple # Prints complete tuple
print tuple[0] # Prints first element of the tuple
print tuple[1:3] # Prints elements starting from 2nd till 3rd
print tuple[2:] # Prints elements starting from 3rd element
print tinytuple * 2 # Prints tuple two times
print tuple + tinytuple # Prints concatenated tuple
```

```
#!/usr/bin/python
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tuple[2] = 1000 # Invalid syntax with tuple
list[2] = 1000 # Valid syntax with list
```

Dictionary

Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.
Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

```
#!/usr/bin/python
dict = {}
dict['one'] = "This is one"
dict[2] = "This is two"
tinydict = {'name': 'john','code':6734, 'dept': 'sales'}
print dict['one'] # Prints value for 'one' key
print dict[2] # Prints value for 2 key
print tinydict # Prints complete dictionary
print tinydict.keys() # Prints all the keys
print tinydict.values() # Prints all the values
```

Basic Operators

Arithmetic Operators

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
// Floor Division	The division of operands where the result is the quotient in which the digits after the decimal point are removed.	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

a=10
b=20

if one of the operands is negative, the result is floored, i.e., rounded away from zero

Practice Exercise 1

Calculate the expression 1-5 in python

1. 7845+2345
2. 89.23-45.2
3. 56x72
4. 9846/47
5. 3^{32}

Comparison Operators

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!= or <>	If values of two operands are not equal, then condition becomes true.	(a<>b) is true
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

a=10
b=20

Assignment Operators

Operator	Example and description
=	c = a + b assigns value of a + b into c
+=	c += a is equivalent to c = c + a
-=	c -= a is equivalent to c = c - a
*=	c *= a is equivalent to c = c * a
/=	c /= a is equivalent to c = c / a
%=	c %= a is equivalent to c = c % a
**=	c **= a is equivalent to c = c ** a
//=	c //= a is equivalent to c = c // a

Membership Operators

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

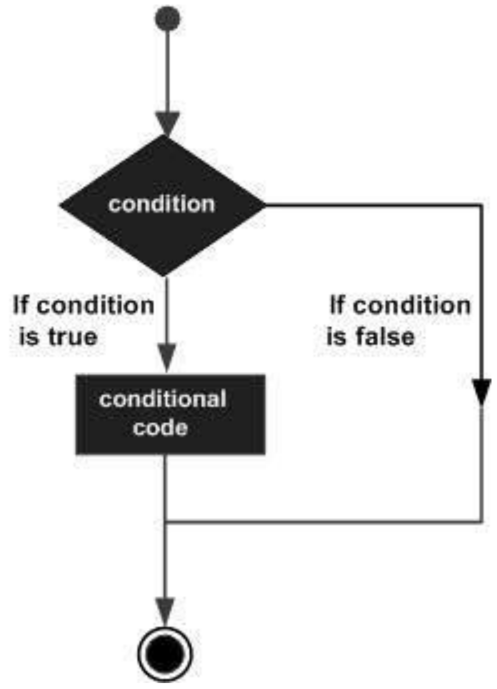
Identity Operators

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here **is not** results in 1 if id(x) is not equal to id(y).

Decision making

An if statement valuates an expression, and executes a group of statements when the expression is true.

if statements
if….else statements
nested statements



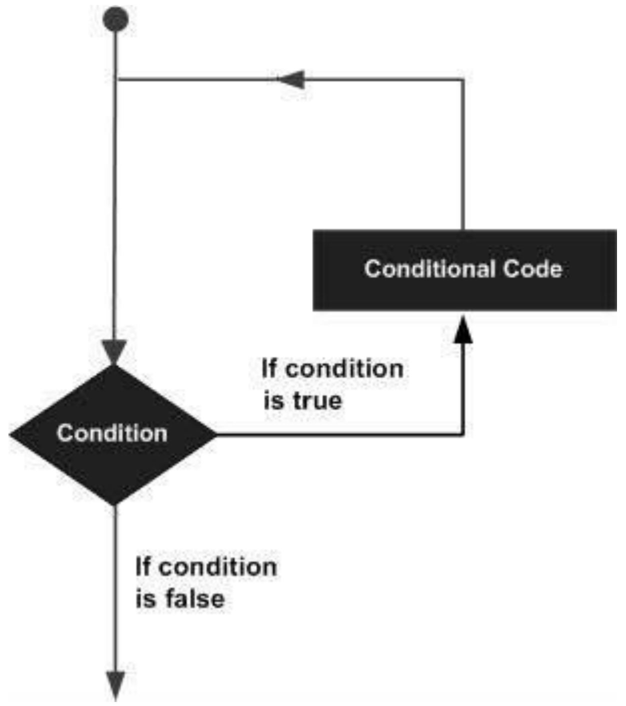
```
#!/usr/bin/python
var = 100
if ( var == 100 ) :
   print "Value of expression is 100"
print "Good bye!"
```

Loops

A loop statement allows us to execute a statement or group of statements multiple times.

Loop Types

- While loop
- For loop
- Nested loops



Control statement

- Break
- continue
- pass

for loop

Executes a group of statements in a loop for a specified number of times.

```
for i in range(start, stop, step):
    Statements
```

```
for i in range(5):
     print i
for j in range(20,30):
    print j
for k in range(1,10,2):
    print k
```

```
#!/usr/bin/env python
a = 1
b = 1
for c in range(1,10):
    print (a)
    n = a + b
    a = b
    b = n
print ("")
```

while loop

syntax

```
while condition:
    statements;
```

Example

```
#!/usr/bin/python
a = 0
while a < 5:
    a += 1 # Same as a = a + 1
    print (a)
```

To add number from user input having a check condition

```
#!/usr/bin/python
a = 1
s = 0
print ('Enter Numbers to add to the sum.')
print ('Enter 0 to quit.')
while a != 0:
    print ('Current Sum: ', s)
    a = raw_input('Number? ')
    a = float(a)
    s += a
print ('Total Sum = ',s)
```

Practice Exercise 2

Write a program to print star pyramid output as shown in figure.

```
    *
   ***
  *****
 *******
*********
```

Function

```
def function_name( parameters ):
        "function_docstring"
        function_suite
        return [expression]
```

syntax

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

- Function blocks begin with the keyword **def** followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Example

```
def printme( str ):
   "This prints a passed string into this function"
   print str
   return
```

Calling a Function

```
#!/usr/bin/python
# Function definition is here
def printme( str ):
   "This prints a passed string into this function"
   print str
   return;
# Now you can call printme function
printme("I'm first call to user defined function!")
printme("Again second call to the same function")
```

Pass by reference

```
#!/usr/bin/python
# Function definition is here
def changeme( mylist ):
   "This changes a passed list into this function"
   mylist.append([1,2,3,4]);
   print "Values inside the function: ", mylist
   return
# Now you can call changeme function
mylist = [10,20,30];
changeme( mylist );
print "Values outside the function: ", mylist
```

Function Arguments:

Required Arguments

```
#!/usr/bin/python
# Function definition is here
def printme( str ):
   "This prints a passed string into this function"
   print str
   return;
# Now you can call printme function
printme(str="My String")
```

Keyword arguments

```
#!/usr/bin/python
# Function definition is here
def printinfo( name, age ):
   "This prints a passed info into this function"
   print "Name: ", name
   print "Age ", age
   return;
# Now you can call printinfo function
printinfo( age=50, name="miki" )
```

Order is not important

Default arguments

```
#!/usr/bin/python
# Function definition is here
def printinfo( name, age = 35 ):
   "This prints a passed info into this function"
   print "Name: ", name
   print "Age ", age
   return;
# Now you can call printinfo function
printinfo( age=50, name="miki" )
printinfo( name="miki" )
```

Variable-length arguments

```
#!/usr/bin/python
# Function definition is here
def printinfo( arg1, *vartuple ):
   "This prints a variable passed arguments"
   print "Output is: " print arg1
   for var in vartuple:
      print var
   return;
# Now you can call printinfo function
printinfo( 10 )
printinfo( 70, 60, 50 )
```

Anonymous Functions

These function are not declared in standard manner.
Use the *lambda* keyword to create small anonymous functions.

Syntax

```
lambda [arg1 [,arg2,.....argn]]:expression
```

Example

```
#!/usr/bin/python
# Function definition is here
sum = lambda arg1, arg2: arg1 + arg2;
# Now you can call sum as a function
print "Value of total : ", sum( 10, 20 )
print "Value of total : ", sum( 20, 20 )
```

- Lambda forms can take any number of arguments but return just one value in the form of an expression. They cannot contain commands or multiple expressions.

Practice Exercise 3

1. Write a function to calculate Fibonacci series?

String revisited

Strings are most popular types in Python. We can create them by enclosing character in quotes.
Python treats single(‘ ’) and double(“ ”) quotes the same.
e.g.
var1=“Hello World!”
var2=‘Python Programming’
Accessing values in string
Python doesn’t support character type; these are treated as strings of length 1.

```
#!/usr/bin/python
var1 = 'Hello World!'
var2 = "Python Programming"
print "var1[0]: ", var1[0]
print "var2[1:5]: ", var2[1:5]
```

Updating String:
We can update an existing string by reassigning a variable to another string. We cannot update an string location wise.

Strings revisited

Common String Methods in Python

Methods	Description
stringVar.count('x')	counts the number of occurrences of 'x' in stringVar
stringVar.find('x')	returns the position of character 'x'
stringVar.lower()	returns the stringVar in lowercase (this is temporary)
stringVar.upper()	returns the stringVar in uppercase (this is temporary)
stringVar.replace('a', 'b')	replaces all occurrences of a with b in the string
stringVar.strip()	removes leading/trailing white space from string

Triple quotes:
It can be used to give multiple line comments.
It is basically used when we want to allow strings to span multiple lines, including verbatim NEWLINEs, TABs, and other special characters.
Syntax consists of three consecutive single or double quotes.
e.g.

```
#!/usr/bin/python

para_str = """this is a long string that is made up of several lines and non-printable characters such as
TAB ( \t ) and they will show up that way when displayed. NEWLINEs within the string, whether
explicitly given like this within the brackets [ \n ], or just a NEWLINE within the variable assignment
will also show up. """

print para_str
```

Lists revisited

The list is a most versatile data type available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values between square brackets.

Example

```
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"]
```

Accessing Values in Lists

Use Square Bracket ([]) for slicing along with the index.

Example:

```
#!/usr/bin/python
List_1=[‘physics’, ‘chemistry’, 1997, 2000]
List_2= [1,2,3,4,5,6,7]
print “list1[0]:”, List_1[0]
print “list2[1:5]”,List_2[1:5]
```

Updating List

We can update single or multiple entries in a list by giving slice on the left hand of the assignment operator

Example:

```
#!/usr/bin/python
list = ['physics', 'chemistry', 1997, 2000];
print "Value available at index 2 : "
print list[2]
list[2] = 2001;
print "New value available at index 2 : " print list[2]
```

Deleting List elements:
To remove a list element we can use del statement if you know the item location or we can use remove statement to remove an item from the list.
Example:

```
#!/usr/bin/python
aList = [123, 'xyz', 'zara', 'abc', 'xyz'];
aList.remove('xyz');
print "List : ", aList
aList.remove('abc');
print "List : ", aList
```

```
#!/usr/bin/python
list1 = ['physics', 'chemistry', 1997, 2000];
print list1
del list1[2];
print "After deleting value at index 2 : "
print list1
```

Basic List operation

Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: print x,	1 2 3	Iteration

Indexing and Slicing

Because lists are sequences, indexing and slicing work the same way for lists as they do for strings.
Assume:
L=[‘spam’, ‘Spam’, ‘SPAM!’]

Python Expression	Results	Description
L[2]	'SPAM!'	Offsets start at zero
L[-2]	'Spam'	Negative: count from the right
L[1:]	['Spam', 'SPAM!']	Slicing fetches sections

Built-in functions and methods

SN	Function with Description
1	cmp(list1, list2) Compares elements of both lists.
2	len(list) Gives the total length of the list.
3	max(list) Returns item from the list with max value.
4	min(list) Returns item from the list with min value.
5	list(seq) Converts a tuple into list.

SN	Methods with Description
1	list.append(obj) Appends object obj to list
2	list.count(obj) Returns count of how many times obj occurs in list
3	list.extend(seq) Appends the contents of seq to list
4	list.index(obj) Returns the lowest index in list that obj appears
5	list.insert(index, obj) Inserts object obj into list at offset index
6	list.pop(obj=list[-1]) Removes and returns last object or obj from list
7	list.remove(obj) Removes object obj from list
8	list.reverse() Reverses objects of list in place
9	list.sort([func]) Sorts objects of list, use compare func if given

Practice Exercise 4

1. Find the max and min value in the following list.
List_1=[23,1,34,79,45,56,88,5,90]
2. Compare two user given list.
3. Reverse a user given list.
4. Sort a user given list.
5. Convert a tuple in a list.

Dictionary Revisited

Accessing values in a dictionary:
To access dict elements, use [] with key to obtain its value. e.g.

```
#!/usr/bin/python
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print "dict['Name']: ", dict['Name']
print "dict['Age']: ", dict['Age']
```

Updating Dictionary:
You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below

```
#!/usr/bin/python
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
dict['Age'] = 8; # update existing entry
dict['School'] = "DPS School"; # Add new entry
print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

Deleting Dictionary elements:
You can either remove individual dict elements or clear an entire dict. E.g.

```
#!/usr/bin/python
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
del dict['Name']; # remove entry with key 'Name'
dict.clear(); # remove all entries in dict
del dict ; # delete entire dictionary
print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

Built in Dictionary function:

SN	Function with Description
1	cmp(dict1, dict2) Compares elements of both dict.
2	len(dict) Gives the total length of the dictionary. This would be equal to the number of items in the dictionary.
3	str(dict) Produces a printable string representation of a dictionary
4	type(variable) Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.

Dictionary Methods

SN	Methods with Description
1	dict.clear() Removes all elements of dictionary *dict*
2	dict.copy() Returns a shallow copy of dictionary *dict*
3	dict.fromkeys() Create a new dictionary with keys from seq and values *set* to *value*.
4	dict.get(key, default=None) For *key* key, returns value or default if key not in dictionary
5	dict.has_key(key) Returns *true* if key in dictionary *dict*, *false* otherwise
6	dict.items() Returns a list of *dict*'s (key, value) tuple pairs
7	dict.keys() Returns list of dictionary dict's keys
8	dict.setdefault(key, default=None) Similar to get(), but will set dict[key]=default if *key* is not already in dict
9	dict.update(dict2) Adds dictionary *dict2*'s key-values pairs to *dict*
10	dict.values() Returns list of dictionary *dict*'s values

Examples

```
#!/usr/bin/python
dict = {'Name': 'Zara', 'Age': 7};
print "Variable Type : %s" % type (dict)
```

```
#!/usr/bin/python
dict1 = {'Name': 'Zara', 'Age': 7};
dict2 = {'Name': 'Mahnaz', 'Age': 27};
dict3 = {'Name': 'Abid', 'Age': 27};
dict4 = {'Name': 'Zara', 'Age': 7};
print "Return Value : %d" % cmp (dict1, dict2)
print "Return Value : %d" % cmp (dict2, dict3)
print "Return Value : %d" % cmp (dict1, dict4)
```

```
#!/usr/bin/python
seq = ('name', 'age', 'sex')
dict = dict.fromkeys(seq)
print "New Dictionary : %s" % str(dict)
dict = dict.fromkeys(seq, 10)
print "New Dictionary : %s" % str(dict)
```

Input and Output

Python has two functions designed for accepting data directly from user:

- input()
- raw_input()

raw_input() asks the user for a string of data (ended with a newline), and simply returns the string. It can also take an argument, which is displayed as a prompt before the user enters the data. E.g.

```
x = raw_input('What is your name?')
print ('Your name is ' + x)
```

input() uses raw_input to read a string of data, and then attempts to evaluate it as if it were a Python program, and then returns the value that results.
Basically, it takes formatted input such as list, dictionary, equation etc.
For example:

```
x = input('What are the first 10 perfect squares? ')
What are the first 10 perfect squares?
map(lambda x: x*x, range(10))
```

Message displayed on screen

User input for evaluation

Turning the strings returned from raw_input() into python types using an idiom such as:

```
x = None
while not x:
   try:
      x = int(raw_input())
   except ValueError:
      print 'Invalid Number'
```

Assignment to be submitted

1. Create a program to count by prime numbers. Ask the user to input a number, then print each prime number up to that number.
2. Instruct the user to pick an arbitrary number from 1 to 100 and proceed to guess it correctly within seven tries. After each guess, the user must tell whether their number is higher than, lower than, or equal to your guess.

Hint: use binary search. Make an array or list of 100 numbers.

3. Write a program to reverse the ordering of words in a string.

e.g. Hello world, this is python!

'python! is this world, Hello‘

Hint: use string function

split() and join()

or you can write it using an array of char

Please submit you assignment to gaurav-vishal@edu.brain.kyutech.ac.jp with subject “Assignment for PHIIP” file name should be your student_number also please write your name in the scripts.