Department of Computer Science

Final Exam Summer 2020

Subject: Data Sciences Time: 9 Am to 12:00 Am

BS (CS,SE) Instructor: M.Ayub Khan

Note:

At the top of the answer sheet there must be the ID, Name and semester of the concerned Student.

Students must have to provide the output of their respective programs. Students have same answers or programs will be considered fail. Programs in Python and codes should be explained clearly.

As this assignment is online so incase of any ambiguity my Whatsapp no. is 03449121116.

Name : Amad Afridi

ID : 13119

Class : BS SE B

Last Semester

Q1. a. What are variables in python explain with help of Python coded examples?

Ans1(a): **VARIABLE IN PYTHON:**

A Python variable is a reserved memory location to store values. In other words, a variable in a python program gives data to the computer for processing. Every value in Python has a datatype. Different data types in Python are Numbers, List,

Tuple , Strings, Dictionary, etc. Variables can be declared by any name or even alphabets like a, aa , abc , etc

How to Declare and use a Variable

Let see an example. We will declare variable "a" and print it.

```
a=100
print (a)
```

Python Example

```
# Declare a variable and initialize it
f = 0
print(f)
# re-declaring the variable works
f = 'guru99'
print(f)
```

➢ Concatenate Variables

whether you can concatenate different data types like string and number together. For example, we will concatenate "Guru" with the number "99".

Unlike Java, which concatenates number with string without declaring number as string, Python requires declaring the number as string otherwise it will show a TypeError

For the following code, you will get undefined output -

```
a="Guru"
b = 99
print a+b
```

Once the integer is declared as string, it can concatenate both "Guru" + **str**("99")= "Guru99" in the output.

```
a="Guru"
b = 99
print(a+str(b))
```

➢ Local & Global Variables

In Python when you want to use the same variable for rest of your program or module you declare it a global variable, while if you want to use the variable in a specific function or method, you use a local variable

Python Example

```
# Declare a variable and initialize it
f = 101
print f
# Global vs. local variables in functions
def someFunction():
# global f
   f = 'I am learning Python'
   print f
someFunction()
print f
```

➢ Delete a variable

You can also delete variable using the command **del** "variable name".

In the example below, we deleted variable f, and when we proceed to print it, we get error "**variable name is not defined**" which means you have deleted the variable.

```
f = 11;
print(f)
del f
print(f)
```

b. What are the rules to define a variable in python?

Ans1(b): **Rules for Python variables:**

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Example

```
#Legal variable names:
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"

#Illegal variable names:
2myvar = "John"
my-var = "John"
my var = "John"
```

Q2. a. What are data types, how many data types are used in python explain with the help of Python coded examples ?

Ans2(a): **Python Data Types**

A Data Type describes the characteristic of a variable.

Python has six standard Data Types:

- Numbers
- String
- List
- Tuple
- Set
- Dictionary

➢ Numbers

In Numbers, there are mainly 3 types which include Integer, Float, and Complex.

These 3 are defined as a class in python. In order to find to which class the variable belongs to you can use type () function.

Example:

```
a = 5
print(a, "is of type", type(a))
```

Output: 5 is of type <class 'int'>

```
b = 2.5
```

```
print(b, "is of type", type(b))
```

Output: 2.5 is of type <class 'float'>

```
c = 6+2j
print(c, "is a type", type(c))
```

Output: (6+2j) is a type <class 'complex'>

- String

A string is an ordered sequence of characters.

We can use single quotes or double quotes to represent strings. Multi-line strings can be represented using triple quotes, ''' or """.

Strings are immutable which means once we declare a string we can't update the already declared string.

Example:

```
Single = 'Welcome'
or
Multi = "Welcome"
```

Multiline: "Python is an interpreted high-level programming language for general-purpose programming.

or

‘’’Python is an interpreted high-level programming language for general-purpose programming We can perform several operations in strings like Concatenation, Repetition, and Slicing.

Concatenation: It means the operation of joining two strings together.

Example:

```
String1 = "Welcome"
String2 ="To Python"
print(String1+String2)
```

Output: Welcome To Python

- List

A list can contain a series of values.

List variables are declared by using brackets [ ]. A list is mutable, which means we can modify the list.

Example:

```
List = [2,4,5.5,"Hi"]
print("List[2] = ", List[2])
```

Output: List[2] = 5.5

- Tuple

A tuple is a sequence of Python objects separated by commas.

Tuples are immutable, which means tuples once created cannot be modified. Tuples are defined using parentheses ().

Example:

```
Tuple = (50,15,25.6,"Python")
print("Tuple[1] = ", Tuple[1])
```

Output: Tuple[1] = 15

- **Set**

A set is an unordered collection of items. Set is defined by values separated by a comma inside braces { }

Example:

```
Set = {5,1,2.6,"python"}
print(Set)
```

Output: {'python', 1, 5, 2.6}

In the set, we can perform operations like union and intersection on two sets.

We can perform Union operation by Using | Operator.

Example:

```
A = {'a', 'c', 'd'}
B = {'c', 'd', 2 }
print('A U B =', A| B)
```

Output: A U B = {'c', 'a', 2, 'd'}

- Dictionary

Dictionaries are the most flexible built-in data type in python.

Dictionaries items are stored and fetched by using the key. Dictionaries are used to store a huge amount of data. To retrieve the value we must know the key. In Python, dictionaries are defined within braces {}.

We use the key to retrieve the respective value. But not the other way around.

Syntax:

Key:value

Example:

```
Dict = {1:'Hi',2:7.5, 3:'Class'}

print(Dict)
```

Output: {1: 'Hi', 2: 7.5, 3: 'Class'}

b. Write a program in python in which integer value is changing into string data type as well as explain in detail.

Ans2(b):**Python Integer to String:**

The str() method allows you to convert an integer to a string in Python. The syntax for this method is

```
str(number_to_convert)
```

Let's walk through an example to show how this method works. In our earlier example, we converted the user's age to an integer using this code

```
raw_user_age = input("What is your age?")

user_age = int(raw_user_age)
```

Let’s say that we want to print a message with our user’s age to the console. We could do this by using the following code

```
print("Your age is: " + user_age)
```

Our code returns an error;

```
Traceback (most recent call last):
  File "main.py", line 3, in <module>
        print("Your age is: " + user_age)
TypeError: can only concatenate str (not "int") to str
```

This is because you cannot concatenate a string to an integer like we have tried to do above. To make our code work, we’re going to have to convert our user’s age to a string. We can do this by using the `str()` method

```
raw_user_age = input("What is your age?")

user_age = int(raw_user_age)

as_string = str(user_age)


print("Your age is: " + as_string)
```

Our code returns:

```
What is your age?
12
```

Your age is: 12

We've successfully converted our integer to a string using str(). Both values are now strings, which means that we can now concatenate the message Your age is: with the user's age.

Conclusion

The int() method is used to convert a string to an integer in Python. This can be useful if you need to store a value as an integer or perform mathematical operations on a value stored as a string. The str() method is used to convert an integer to a string.

Q3. Why print() and type functions are used in python explain with the help of

python coded examples for each function and explain in detail as well ?

Ans3: Python print() Function

Definition and Usage

The print() function prints the specified message to the screen, or other standard output device.

The message can be a string, or any other object, the object will be converted into a string before written to the screen.

Syntax

```
print(object(s), sep=separator, end=end, file=file, flush=flush)
```

More Examples

Example

Print more than one object:

```
print("Hello", "how are you?")
```

Example

Print a tuple:

```
x = ("apple", "banana", "cherry")
print(x)
```

Example

Print two messages, and specify the separator:

```
print("Hello", "how are you?", sep="---")
```

Python | type() function

type() method returns class type of the argument(object) passed as parameter. type() function is mostly used for debugging purposes. Two different types of arguments can be passed to type() function, single and three argument. If single argument type(obj) is passed, it returns the type of given object. If three arguments type(name, bases, dict) is passed, it returns a new type object.

Syntax :

```
type(object)

type(name, bases, dict)
```

Returntype :

returns a new type class or essentially a metaclass.

Example1:

```
# Python3 simple code to explain
# the type() function
print(type([]) is list)

print(type([]) is not list)

print(type(()) is tuple)

print(type({}) is dict)

print(type({}) is not list)
```

Output :

```
True
False
True
True
True
```

Example2:

```
# Python3 code to explain
# the type() function
```

```
# Class of type dict
class DictType:
    DictNumber = {1:'John', 2:'Wick',
                  3:'Barry', 4:'Allen'}

    # Will print the object type
    # of existing class
    print(type(DictNumber))

# Class of type list
class ListType:
    ListNumber = [1, 2, 3, 4, 5]

    # Will print the object type
    # of existing class
    print(type(ListNumber))

# Class of type tuple
class TupleType:
    TupleNumber = ('Geeks', 'for', 'geeks')

    # Will print the object type
    # of existing class
    print(type(TupleNumber))

# Creating object of each class
d = DictType()
l = ListType()
t = TupleType()
```

Output :

```
<class 'dict'>
<class 'list'>
<class 'tuple'>
```

Q4. How addition operator is used to update the values of variables explain with the help of Python coded example as well as explain the program?

Ans4: **Addition operator is used to update the values of variables**

```
x = 6       # initialize x
print(x)
x = x + 1    # update x
print(x)
```

If you try to update a variable that doesn't exist, you get an error because Python evaluates the expression on the right side of the assignment operator before it assigns the resulting value to the name on the left. Before you can update a variable, you have to **initialize** it, usually with a simple assignment. In the above example, x was initialized to 6

Or

```
a = a + 1
```

Since these operations are so common, Python has ﷺshortcutﷺ operators that make typing and understanding easier

+=

These 2 expressions produce the same result:

```
a = a + 5

a += 5
```

Other operators that work the same way include -=, *=, /=

Precedence

Most modern programming languages interpret compound math operations in the same way

They generally follow accepted mathematical practice in formal expressions

In the following expression:

```
>>> 3 + 4 * 5

23
```

The rules of precedence require that a multiplication in an expression be performed before an addition

We can change precedence by using parentheses:

```
>>> (3 + 4) * 5

35
```

The specific Python rules of precedence are here

The numpy module

The Anaconda distribution includes numpy, a general purpose ⍰scientific computing⍰ collection of useful functions

numpy contains many good analytical functions as well as simpler trig and utility functions

To use numpy, your module must import it (I&#65533;m assuming we&#65533;re working in a text editor here):

```
import numpy

sinOneHalf = numpy.sin(.5)

myAngleInDegrees = 180

myAngleInRadians = numpy.deg2rad(myAngleInDegrees)

# and so on&#65533;
```

Here is some code that exercises trig functions and constants in numpy

In the file I frequently use the \ character to indicate that a python statement extends onto the next line in the text

I also use the str() command to convert numerical information to a string

You need to do this if you mix strings and numerical data in a single print statement

Q5. What type of errors do occur in Python, write the a program with different types of errors as well as write separate correction code in python as well as explain the errors?

Ans5: The most common reason of an error in a Python program is when a certain statement is not in accordance with the prescribed usage. Such an error is called a syntax error. The Python interpreter immediately reports it, usually along with the reason.

```
>>> print "hello"
SyntaxError: Missing parentheses in call to 'print'. Did you mean print("hello")?
```

In Python, print is a built-in function and requires parentheses. The statement above violates this usage and hence syntax error is displayed.

Many times though, a program results in an error after it is run even if it doesn't have any syntax error. Such an error is a runtime error, called an exception. A number of built-in exceptions are defined in the Python library. Let's see some common error types.

Logic error

These are the most difficult type of error to find, because they will give unpredictable results and may crash your program. A lot of different things can happen if you have a logic error. However these are very easy to fix as you can use a debugger, which will run through the program and fix any problem

IndexError

is thrown when trying to access an item at an invalid index.

```
>>> L1=[1,2,3]
>>> L1[3]
Traceback (most recent call last):
File "<pyshell#18>", line 1, in <module>
L1[3]
IndexError: list index out of range
```

ModuleNotFoundError

is thrown when a module could not be

```
>>> import notamodule
Traceback (most recent call last):
File "<pyshell#10>", line 1, in <module>
import notamodule
ModuleNotFoundError: No module named 'notamodule
```

KeyError

is thrown when a key is not found.

```
>>> D1={'1':"aa", '2':"bb", '3':"cc"}
>>> D1['4']
Traceback (most recent call last):
File "<pyshell#15>", line 1, in <module>
D1['4']
KeyError: '4'
```

ImportError

```
>>> from math import cube
Traceback (most recent call last):
from math import cube
ImportError: cannot import name 'cube
```

StopIteration

is thrown when the next() function goes beyond the iterator items.

```
>>> it=iter([1,2,3])
>>> next(it)
1
>>> next(it)
```

```
2
>>> next(it)
3
>>> next(it)
Traceback (most recent call last):
File "<pyshell#23>", line 1, in <module>
next(it)
```

is thrown when a specified function can not be found.

`StopIteration`

TypeError

is thrown when an operation or function is applied to an object of an inappropriate type.

```
>>> '2'+2
Traceback (most recent call last):
File "<pyshell#23>", line 1, in <module>
'2'+2
TypeError: must be str, not int
```

ValueError

is thrown when a function's argument is of an inappropriate type.

```
>>> int('xyz')
Traceback (most recent call last):
File "<pyshell#14>", line 1, in <module>
int('xyz')
ValueError: invalid literal for int() with base 10: 'xyz
```

NameError

is thrown when an object could not be found.

```
>>> age
Traceback (most recent call last):
File "<pyshell#6>", line 1, in <module>
age
NameError: name 'age' is not define
```

ZeroDivisionError

is thrown when the second operator in the division is zero

```
>>> x=100/0
Traceback (most recent call last):
File "<pyshell#8>", line 1, in <module>
x=100/0
ZeroDivisionError: division by zer
```

KeyboardInterrupt

is thrown when the user hits the interrupt key (normally Control-C) during the execution of the program.

```
>>> name=input('enter your name')
enter your name^c
Traceback (most recent call last):
File "<pyshell#9>", line 1, in <module>
name=input('enter your name')
KeyboardInterrupt
```

Correcting Coding Errors in Python

When coding programs there are three common types of error that can occur. It is useful to recognize these different error types in Python programming so they can be corrected more easily:

- **Syntax Error** – occurs when the interpreter encounters code that does not conform to the Python language rules. For example, a missing quote mark around a string. The interpreter halts and reports the error without executing the program.
- **Runtime Error** – occurs during execution of the program, at the time when the program runs. For example, when a variable name is later mis-typed so the variable cannot be recognized. The interpreter runs the program but halts at the error and reports the nature of the error as an "Exception".
- **Semantic Error** – occurs when the program performs unexpectedly. For example, when order precedence has not been specified in an expression. The interpreter runs the program and does not report an error.

Correcting syntax and runtime errors is fairly straightforward, as the interpreter reports where the error occurred or the nature of the error type, but semantic errors require code examination.

Step 1
Open an IDLE Edit Window then add a statement to output a string that omits a closing quote mark
print('Coding for Beginners in easy steps)

Step 2
Save then run the program to see the interpreter highlight the syntax error and indicate its nature

The red syntax error indicator points to the line where the End Of Line (EOL) error occurs.

Step 3

Insert a quote mark before the closing parenthesis to terminate the string and save then run the program again – to see the error has been corrected

Step 4

Next, begin a new program by initializing a variable then try to output its value with an incorrect variable name – to see the interpreter report a runtime error

```
title = 'Coding for Beginners in easy steps'
print( titel )
```

Step 5

Amend the variable name to match that in the variable declaration and save then run the program again – to see the error has been corrected

Step 6

Now, begin a new program by initializing a variable then try to output an expression using its value without explicit precedence – to see a possibly unexpected result of 28

```
num = 3
print( 'Result: ' , num * 8 + 4 )
```

Step 7

Add parentheses to group the expression as **3 * (8 + 4)** then save the file and run the program again – to see the expected result of 36, correcting the semantic error