NAME: HASSAN KAMAL

ID#: 12925

SUBJECT NAME: DATA SCIENCES

QUESTION#1:

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

ANSWER:

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.

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

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

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

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

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

ANSWER:

- Variables names must start with a letter or an underscore, such as:

    - _underscore
    - underscore_
- The remainder of your variable name may consist of letters, numbers and underscores.
    - password1
    - n00b
    - un*der*scores
- Names are case sensitive.
    - case*sensitive, CASE*SENSITIVE, and Case_Sensitive are each a different variable.

- 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)

QUESTION#2:

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

ANSWER:

a data type or simply type is an attribute of data which tells the compiler or interpreter how the programmer intends to use the data.

Python has six standard Data Types:

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

Numbers

- Integers, floating point numbers and complex numbers fall under Python numbers category. They are defined as int, float and complex classes in Python.
- We can use the type() function to know which class a variable or a value belongs to. Similarly, the is instance() function is used to check if an object belongs to a particular class.

  PYTHON CODE:
- a = 5
- print(a, "is of type", type(a))
-
- a = 2.0
- print(a, "is of type", type(a))
-
- a = 1+2j
- print(a, "is complex number?", is instance(1+2j,complex))

  Strings
- String is sequence of Unicode characters. We can use single quotes or double quotes to represent strings. Multi-line strings can be denoted using triple quotes, ''' or """.

  PYTHON CODE:

- s = "This is a string"
- print(s)
- s = '''A multiline
- string'''
- print(s)

List

- List is an ordered sequence of items. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type.
- Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets [ ].

  PYTHON CODE:
- a = [1, 2.2, 'python']
- We can use the slicing operator [ ] to extract an item or a range of items from a list. The index starts from 0 in Python.
- a = [5,10,15,20,25,30,35,40]

- # a[2] = 15
- print("a[2] = ", a[2])
-
- # a[0:3] = [5, 10, 15]
- print("a[0:3] = ", a[0:3])

Tuple

- Tuple is an ordered sequence of items same as a list. The only difference is that tuples are immutable. Tuples once created cannot be modified.
- Tuples are used to write-protect data and are usually faster than lists as they cannot change dynamically.
- It is defined within parentheses () where items are separated by commas.
  PYTHON CODE:
- t = (5,'program', 1+3j)
- We can use the slicing operator [] to extract items but we cannot change its value.
- t = (5,'program', 1+3j)
-
- # t[1] = 'program'
- print("t[1] = ", t[1])
-
- # t[0:3] = (5, 'program', (1+3j))
- print("t[0:3] = ", t[0:3])
-
- # Generates error
- # Tuples are immutable
- t[0] = 10

- # a[5:] = [30, 35, 40]
- print("a[5:] = ", a[5:])

Set

- Set is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }. Items in a set are not ordered.
  PYTHON CODE:
- a = {5,2,3,1,4}
-
- # printing set variable
- print("a = ", a)

- # data type of variable a
- print(type(a))
- **Dictionary**
- Dictionary is an unordered collection of key-value pairs.
- It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value.
- In Python, dictionaries are defined within braces {} with each item being a pair in the form key:value. Key and value can be of any type.

PYTHON CODE

- >>> d = {1:'value','key':2}
- >>> type(d)
- <class 'dict'>
- We use key to retrieve the respective value. But not the other way around.
- d = {1:'value','key':2}
- print(type(d))
-
- print("d[1] = ", d[1]);
-
- print("d['key'] = ", d['key']);
-
- # Generates error
- print("d[2] = ", d[2]);

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

ANSWER:

You can use the "+" and "*" operators on strings.

- "Adding" character strings concatenates them.

```
full_name = 'Ahmed' + ' ' + 'Walsh'
print(full_name)
```

Ahmed Walsh

- Multiplying a character string by an integer *N* creates a new string that consists of that character string repeated *N* times.
    - Since multiplication is repeated addition.

```
separator = '=' * 10
print(separator)
```

QUESTION#3:
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 ?
ANSWER:

Print function is used in python because if you want to print a number , name , or anything it is printed
For example:

```
a, b, c = 5, 3.2, "Hello"

print (a)
print (b)
print (c)
```

type function is used in python because if you want add any type you can type it like int, float, double, etc.

```
a = 0b1010 #Binary Literals
b = 100 #Decimal Literal
c = 0o310 #Octal Literal
d = 0x12c #Hexadecimal Literal

#Float Literal
float_1 = 10.5
float_2 = 1.5e2

#Complex Literal
x = 3.14j

print(a, b, c, d)
print(float_1, float_2)
print(x, x.imag, x.real)
```

QUESTION#4:

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?

ANSWER:

```
        a_number = 5          # a_number becomes 5
a_number = total        # a_number becomes the value of total
a_number = total + 5    # a_number becomes the value of total + 5
a_number = a_number + 1  # a_number becomes the value of a_number + 1
```

The last statement might look a bit strange if we were to interpret = as a mathematical equals sign – clearly a number cannot be equal to the same number plus one! Remember that = is an assignment operator – this statement is assigning a *new* value to the variable a_number which is equal to the *old* value of a_number plus one.

Assigning an initial value to variable is called *initialising* the variable. In some languages defining a variable can be done in a separate step before the first value assignment. It is thus possible in those languages for a variable to be defined but not have a value – which could lead to errors or unexpected behavior if we try to use the value before it has been assigned. In Python a variable is defined and assigned a value in a single step, so we will almost never encounter situations like this.

The left-hand side of the assignment statement must be a valid target:

```
# this is fine:
a = 3

# these are all illegal:
3 = 4
3 = a
a + b = 3
```

An assignment statement may have multiple targets separated by equals signs. The expression on the right-hand side of the last equals sign will be assigned to all the targets. All the targets must be valid:

```
# both a and b will be set to zero:
a = b = 0
```

```
# this is illegal, because we can't set 0 to b:
a = 0 = b
```

QUESTION#5:

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

ANSWER:

Errors or mistakes in a program are often referred to as bugs. They are almost always the fault of the programmer. The process of finding and eliminating errors is called debugging. Errors can be classified into three major groups:

- Syntax errors
- Runtime errors
- Logical errors

Syntax errors:

Python will find these kinds of errors when it tries to parse your program and exit with an error message without running anything. Syntax errors are mistakes in the use of the Python language and are analogous to spelling or grammar mistakes in a language like English: for example, the sentence *Would you some tea?* does not make sense – it is missing a verb.

Here are some examples of syntax errors in Python:

```
myfunction(x, y):
   return x + y

else:
   print("Hello!")

if mark >= 50
   print("You passed!")

if arriving:
```

```
   print("Hi!")
esle:
   print("Bye!")

if flag:
print("Flag is set!")
```

Runtime errors:

If a program is syntactically correct – that is, free of syntax errors – it will be run by the Python interpreter. However, the program may exit unexpectedly during execution if it encounters a *runtime error* – a problem which was not detected when the program was parsed but is only revealed when a particular line is executed. When a program comes to a halt because of a runtime error, we say that it has crashed.

Consider the English instruction *flap your arms and fly to Australia.* While the instruction is structurally correct and you can understand its meaning perfectly, it is impossible for you to follow it.

Some examples of Python runtime errors:

- division by zero
- performing an operation on incompatible types
- using an identifier which has not been defined
- accessing a list element, dictionary value or object attribute which doesn't exist
- trying to access a file which doesn't exist

Logical errors are the most difficult to fix. They occur when the program runs without crashing, but produces an incorrect result. The error is caused by a mistake in the program's logic. You won't get an error message, because no syntax or runtime error has occurred. You will have to find the problem on your own by reviewing all the relevant parts of your code – although some tools can flag suspicious code which looks like it could cause unexpected behaviour.

Sometimes there can be absolutely nothing wrong with your Python implementation of an algorithm – the algorithm itself can be incorrect. However,

more frequently these kinds of errors are caused by programmer carelessness. Here are some examples of mistakes which lead to logical errors:

- using the wrong variable name
- indenting a block to the wrong level
- using integer division instead of floating-point division
- getting operator precedence wrong
- making a mistake in a Boolean expression
- off-by-one, and other numerical errors.