Final Term

Submitted By:

Hamza khan afridi (13071)

BS (SE)

Subject :

Data Sciences

Question1 (a): What are variables in python explain with help of Python coded examples?

Answer: Variables are containers for storing data values. Python has no command for declaring a variable. A variable is created the moment you first assign a value to it. 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

Example:

```
x = 5

y = "John"

print(x)

print(y)
```

Example: Variables do not need to be declared with any particular *type*, and can even change type after they have been set.

```
x = 4 # x is of type int

x = "Sally" # x is now of type str

print(x)
```

(b): What are the rules to define a variable in python?

Answer: There are some rules to follow:

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

```
_hamza

hamza_
```

2) The remainder of your variable name may consist of letters, numbers and underscores.

```
password1

Pr0
```

Underscores

3) Names are case sensitive.

casesensitive, CASESENSITIVE, and Case_Sensitive are each a different variable.

4) Readability is very important.

Hamza _Afridi

Hamza afridi

Question2 (a): What are data types, how many data types are used in python explain with the help of Python coded examples ?

Answer: In Python, data types are used to classify one particular type of data. This is important because the specific data type you use will determine what values you can assign to it and what you can do to it.

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.

Example:

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

2) 2.5
print(b, "is of type", type(b))

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

String: A string is an ordered sequence of characters.
Example:
String1 = "Welcome"
String2 ="To Python"
print(String1+String2)

List: A list can contain a series of values.
Example:
List = [2,4,5.5,"Hi"]
print("List[2] = ", List[2])

Tuple: A tuple is a sequence of Python objects separated by commas.
Example:
Tuple = (50,15,25.6,"Python")
print("Tuple[1] = ", Tuple[1])

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)

Dictionary: Dictionaries are the most flexible built-in data type in python.
Example:
Dict = {1:'Hi',2:7.5, 3:'Class'}
print(Dict)

(b): Write a program in python in which integer value is changed in to string data type as well as explain in detail.

Answer:

num_int = 123

```
num_str = "456"
print("Data type of num_int:",type(num_int))
print("Data type of num_str before Type Casting:",type(num_str))
num_str = int(num_str)
print("Data type of num_str after Type Casting:",type(num_str))
num_sum = num_int + num_str
print("Sum of num_int and num_str:",num_sum)
print("Data type of the sum:",type(num_sum))
```

Question3: 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: 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.

```
int_list = [1,2,3,4,5,6]
print(int_list)
```

You've seen that print() is a function in Python 3. More specifically, it's a built-in function, which means that you don't need to import it from anywhere:

```
>>> print
<built-in function print>
```

The 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

```
class DictType:
```

```
 DictNumber = {1:'John', 2:'Wick',  3:'Barry', 4:'Allen'}
 print(type(DictNumber))
class ListType:
 ListNumber = [1, 2, 3, 4, 5]
print(type(ListNumber))
class TupleType:
TupleNumber = ('Geeks', 'for', 'geeks')
print(type(TupleNumber))
d = DictType()
l = ListType()
t = TupleType()
```

type() function is basically used for debugging purposes. When used other string functions like .upper(), .lower(), split() with text extracted from a web crawler, it might not work because they might be of different type which doesn't support string functions. And as a result it will keep throwing errors, which are very difficult to debug [Consider the error as : Generator Type has no attribute lower().type() function can be used at that point to determine the type of text extracted and then change it to other forms of string before we use string functions or any other operations on it.

type() with three arguments can be used to dynamically initialize classes or existing classes with attributes. It is also used to register database tables with SQL.

Question4 : How addition operator is used to update the values of variables explain with he help of Python coded example as well as explain the program?

Answer: One of the most common forms of reassignment is an update where the new value of the variable depends on the old.

For example: x = x + 1

This means get the current value of x, add one, and then update x with the new value. The new value of x is the old value of x plus 1. Although this assignment statement may look a bit strange, remember that executing assignment is a two-step process. First, evaluate the right-hand side expression. Second, let the variable name on the left-hand side refer to this new resulting object. The fact that x appears on both sides does not matter. The semantics of the assignment statement makes sure that there is no confusion as to the result. The visualizer makes this very clear.

```
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.Updating a variable by adding 1 is called an increment; subtracting 1 is called a decrement. Sometimes programmers also talk about bumping a variable, which means the same as incrementing it by 1.

Question5 : 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?

Answer: In python there are three types of errors; syntax errors, logic errors and exceptions.

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.

Common Python syntax errors include:

leaving out a keyword

putting a keyword in the wrong place

leaving out a symbol, such as a colon, comma or brackets

misspelling a keyword

incorrect indentation

empty block

Error code:

```
ages = {
    'pam': 24,
    'jim': 24
    'michael': 43
}
print(f'Michael is {ages["michael"]} years old.')
```

Correct Code:

```
ages = {
    'pam': 24,
    'jim': 24,
    'michael': 43,
}
print(f'Michael is {ages["michael"]} years old.')
```

Logical errors: 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

If you misspell an identifier name, you may get a runtime error or a logical error, depending on whether the misspelled name is defined.

Error code:

```
float average(float a, float b)
{
   return a + b / 2;
}
```

Correct Code:

```
float average(float a, float b)
{
   return  (a + b) / 2 ;
}
```

Exception: Exceptions arise when the python parser knows what to do with a piece of code but is unable to perform the action. An example would be trying to access the internet with python without an internet connection; the python

interpreter knows what to do with that command but is unable to perform it. Dealing with exceptions Unlike syntax errors, exceptions are not always fatal. Exceptions can be handled with the use of a try statement.