Mid Semester Assignment

Course Name: Data Sciences

Submitted By:

Inzamam (12998)

BS (SE-8) Section: A

Submitted To:

Sir M.Ayub Khan

Dated: 19th August 2020

Department of Computer Science,
IQRA National University, Peshawar Pakistan

Department of Computer Science

Mid Semester Assignment Summer 2020

Subject: Data Sciences

Time: 90 mins

BS (CS,SE)

Instructor: M.Ayub Khan

There are total **3** questions in this paper.

Max Marks: 30

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.

Each question carry equal marks.
Please answer briefly.

Q1. What type of errors do occur in Python, write the a program with different types of errors as well as write separate correction code?

Q2. What are Boolean String test, write the code for each Boolean string test code?

Q3. What is formatting string input mean in Python, write a program in which formatting string input is used?

Q1

Answer:

Errors:

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 can find these sorts of errors once it tries to dissect your program, and exit with an error message while not running anything. Syntax errors are mistakes within the use of the Python language, and are analogous to spelling or synchronic linguistics mistakes in a language like English: for instance, the sentence Would you some tea? doesn't make sense – it's missing a verb.

Common Python syntax errors include:

- leaving out a keyword
- putting a keyword in the wrong place
- leaving out a symbol, like a colon, comma or brackets
- misspelling a keyword
- Incorrect indentation
- Empty block

Python can do its best to inform you wherever the error is found, however generally its messages are often misleading: for instance, if you forget to escape a quotation mark within a string you will get a syntax error referring to a place later in your code, even supposing that's not the real source of the problem. If you can't see anything wrong on the line laid out in the error message, attempt backtracking through the last few lines. As you program more, you'll improve at distinctive and fixing errors.

Examples of syntax errors in Python are as follows:

```
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's, free of syntax errors – it'll be run by the Python interpreter. However, the program could exit unexpectedly throughout execution if it encounters a run-time error – a problem that wasn't detected when the program was parsed, however is simply unconcealed once a specific line is executed. Once a program comes to a halt due to a run-time error, we say that it's crashed.

Consider the English instruction flap your arms and fly to Australia. Whereas the instruction is structurally correct and you'll perceive its meaning perfectly, it's not possible for you to follow it.

Examples of Python runtime errors are as follows:

- Division by zero
- performing an operation on incompatible varieties
- using an identifier that has not been outlined
- accessing an inventory element, dictionary value or object attribute that doesn't exist
- trying to access a file that doesn't exist

Runtime errors typically sneak in if you don't think about all attainable values that a variable may contain, particularly when you are processing user input. You ought to continuously attempt to add checks to your code to create positive that it will manage bad input and edge cases graciously**.** We can solve these types of error by using exception handlings. In exception handling we can tell python what to do when an error occurs instead if crashing the app

Logical errors:

Logical errors are the foremost tough to fix. They occur once the program runs while not crashing, however produces an incorrect result. The error is caused by a mistake within the program's logic. You won't get an error message, because no syntax or run-time error has occurred. You'll have to be compelled to realize the matter on your own by reviewing all the relevant elements of your code – though some tools will flag suspicious code that seems like it may cause surprising behavior.

Sometimes there are often absolutely nothing wrong along with your Python implementation of an algorithm – the algorithm itself are often incorrect. However, more often these sorts of errors are caused by coder carelessness. Here are some examples of mistakes that result in logical errors:

- using the incorrect variable name
- indenting a block to the incorrect level
- using integer division rather than floating-point division
- getting operator precedence wrong
- making an error in a mathematician expression
- Off-by-one, and different numerical errors

If you misspell an identifier name, you will get a run-time error or a logical error, depending on whether or not the misspelled name is outlined.

A common source of variable name mix-ups and incorrect indentation is frequent repetition and pasting of enormous blocks of code. If you've got several duplicate lines with minor variations, it's terribly straightforward to miss a necessary change after you are editing your pasted lines. You ought to continuously try and factor out excessive duplication using functions and loops.

Q2

Answer:

Booleans:

Boolean values are the 2 constant objects False and True.

They are accustomed to represent truth values (other values also can be considered false or true).

In numeric contexts (for instance, when used as the argument to an arithmetic operator), they behave just like the integers 0 and 1, severally.

The built-in function bool() is used to cast any value to a Boolean, if the value is taken as a truth value. They are written as False and True, severally.

Boolean Strings:

A string in Python can be tested for truth value.

The return type will be in Boolean value (True or False)

Let’s make an example, by first create a new variable and give it a value.

```
my_string = "Hello World"
```

```
my_string.isalnum()          #check if all char are numbers
my_string.isalpha()          #check if all char in the string are alphabetic
my_string.isdigit()          #test if string contains digits
my_string.istitle()          #test if string contains title words
my_string.isupper()          #test if string contains upper case
my_string.islower()          #test if string contains lower case
my_string.isspace()          #test if string contains spaces
my_string.endswith('d')      #test if string endswith a d
my_string.startswith('H')    #test if string startswith H
```

To see what the return value (True or False) will be, simply print it out.

```
my_string="Hello World"
```

```
print my_string.isalnum()           #False
print my_string.isalpha()           #False
print my_string.isdigit()           #False
print my_string.istitle()           #True
print my_string.isupper()           #False
print my_string.islower()           #False
print my_string.isspace()           #False
print my_string.endswith('d')       #True
print my_string.startswith('H')     #True
```

Boolean string tests:

Methods

- .isalpha()
- .isalnum()
- .istitle()
- .isdigit()
- .islower()
- .isupper()
- .startswith()

type **str** has methods that return a Boolean (True or False) for different tests on the properties of stings. >python "Hello".isapha() out:[ ]   True

.isalpha() returns True if all characters in the string ("Hello") are alphabetical, otherwise returns False

Examples:

In [4]:

"Python".isalpha()

Out[4]:

True

In [17]:

"3rd".isalnum()

Out[17]:

True

In [6]:

"A Cold Stromy Night".istitle()

Out[6]:

True

In [7]:

"1003".isdigit()

Out[7]:

```
True
```

In [8]:

```
cm_height = "176"
print("cm height:",cm_height, "is all digits =",cm_height.isdigit())
```

```
cm height: 176 is all digits = True
```

In [9]:

```
print("SAVE".islower())
print("SAVE".isupper())
```

```
False
True
```

In [10]:

```
"Boolean".startswith("B")
```

Out[10]:

```
True
```

test stings with .isalpha()

In [13]:

```
# [ ] Use .isalpha() on the string "alphabetical"
string = "alphabetical"
print('string', string, 'is alphabetical?', string.isalpha())
```

```
string alphabetical is alphabetical? True
```

In [14]:

```
# [ ] Use .isalpha() on the string: "Are spaces and punctuation Alphabetical?"
"Are spaces and punctuation Alphabetical?".isalpha()
```

Out[14]:

```
False
```

In [15]:

```
# [ ] initailize variable alpha_test with input
```

```
alpha_test = input()

# [ ] use .isalpha() on string variable alpha_test

alpha_test.isalpha()

some

Out[15]:

True
```

Q3

Answer:

String Formatting:

Python uses C-style string formatting to create new, formatted strings. The "%" operator is used to format a set of variables enclosed in a "tuple" (a fixed size list), together with a format string, which contains normal text together with "argument specifiers", special symbols like "%s" and "%d".

Here are some basic argument specifiers:

%s - String (or any object with a string representation, like numbers)

%d - Integers

%f - Floating point numbers

%.<number of digits>f - Floating point numbers with a fixed amount of digits to the right of the dot.

%x/%X - Integers in hex representation (lowercase/uppercase)

Example

We write a format string which prints out the data using the following syntax:

Hello John Doe. Your current balance is $53.44.

Input code:

```
data = ("John", "Doe", 53.44)

format_string = "Hello %s %s. Your current balance is $%s."

print(format_string % data)
```

Output: Hello John Doe. Your current balance is $53.44.