| | | |
|---|---|---|
| Faculty of Engineering<br>South Eastern University of Sri Lanka | Course | BSc Eng (Hons) |
| | Semester | 1 |
| | Subject | CS 13001 Introduction to Computing |

SEU/IS/...../EG/.....

Lab Session 3: Conditional Execution

Lab Objectives:

1) Understanding Boolean expressions.
2) Learning logical operators.
3) Writing conditional Python statements.
4) Handling exceptions using the try and except method.
5) Understanding how logical expressions are evaluated in Python.

## 1. Boolean expressions

A Boolean expression is either *true* or *false*. For instance, 5 is greater than 2 is true while 3 is equal to 1 is false. The following table summarises the comparison operators used in Python.

```
x == y      # x is equal to y
x != y      # x is not equal to y
x > y       # x is greater than y
x < y       # x is less than y
x >= y      # x is greater than or equal to y
x <= y      # x is less than or equal to y
x is y      # x is the same as y
x is not y  # x is not the same as y
```

***Exercise 1:*** Try the following Boolean expressions in the Python interpreter and note down the answers on the side.

```
>>> 5 == 5

>>> 5 == 6

>>> 5 >= 6


>>> x = 5

>>> y = 5.0

>>> x == y

>>> x is y
```

## 2. Logical operators

There are three logical operators: *and*, *or*, and *not*. The semantics (meaning) of these operators is similar to their meaning in English.

***Exercise 2:*** Solve the following Boolean expressions that contain logical operators by hand. Verify your answers in the Python interpreter.

a. 5 > 0 and 5 < 10

b. 7 % 2 or 6 % 3

c. 5 >= 2 and not 5 < 1

## 3. Conditional statements

Conditional statements give us the ability to check conditions – i.e. evaluate Boolean logic – and change the behavior of the programme accordingly. For example:

```
if x > 0 :
    print('x is positive')
```

The boolean expression after the if statement is called the condition. We end the if statement with a colon character (:) and the line(s) after the if statement are indented.

***Exercise 3:*** Write the following code as a Python script, save it as ex3.py and run it through the terminal.

```
x = input(‘Enter a number : ‘)
if x > 0:
    print (x , ‘is a positive number!’)
```

***Hint:*** *There is an error in this code. You need to fix it.*

***Exercise 4:*** Execute the same code in the Python interpreter.

```
>>> x = input('Enter a number : ')
Enter a number : 10
>>> if int(x) > 0 :
...     print(x , 'is a positive number!')
...
10 is a positive number!
```

### 4. Alternative execution

Alternative execution is when there are two possibilities and the condition decides which one gets executed. The syntax is as follows:

```
if x % 2 == 0 :
        print(x, ' is even')
else :
        print(x, ' is odd')
```

***Exercise 5:*** Write a Python script that asks the user to enter a number, checks whether a number is positive or negative, and prints out a message to that effect. Consider 0 to be positive to avoid confusion. The programme should run as follows:

Enter a number : 24
24 is positive

***Exercise 6:*** Write a Python script that asks the user to enter a number, checks whether a number is odd or even, and prints out a message to that effect. The programme should run as follows:

Enter a number : 113
113 is odd

### 5. Chained conditionals

When there are more than two possibilities we use chained conditional statements. In addition to if and else we use 'else if'. In Python else if is shortened by elif.

```
if x < y:
    print('x is less than y')
elif x > y:
    print('x is greater than y')
else:
    print('x and y are equal')
```

There is no limit on the number of elif statements. If there is an else clause, it has to be at the end, but there doesn't have to be one.

### 6. Nested conditionals

One conditional statement can be nested within another conditional statement. We could have written the example provided in the previous section in the following manner.

```
if x == y:
    print('x and y are equal')
else:
    if x < y:
        print('x is less than y')
    else:
        print('x is greater than y')
```

***Exercise 7:*** Write a program that asks the user for two numbers, compares the two numbers and tells whether they are equal or which number is greater than the other. Your programme should be able to handle floating point numbers and should run as shown.

Enter first number : 23
Enter second number : -14
23.0 is greater than -14.0

***Exercise 8:*** Re-write the following nested conditional as a single conditional statement by hand.

```
if 0 < x:
    if x < 10:
        print(x, 'is a positive single-digit number.')
```

## 7. Handling exceptions with try and except

When we ask users to input data they are free to enter whatever they want. Your programme might expect a number but the user might enter his name. When this happens Python programme crashes. We would like to avoid this situation.

For example, this programme converts a Fahrenheit temperature to a Celsius temperature. But what if a user enters his name instead of a Fahrenheit temperature value?

```
inp = input('Enter Fahrenheit Temperature: ')
fahr = float(inp)
cel = (fahr - 32.0) * 5.0 / 9.0
print(cel)
```

```
python fahren.py
Enter Fahrenheit Temperature: Nilanka Sankalpa
Traceback (most recent call last):
    File "fahren.py", line 2, in <module>
        fahr = float(inp)
ValueError: could not convert string to float: 'Nilanka Sankalpa'
```

We avoid these unexpected errors by using the “try / except” structure available in Python. We can re write our temperature conversion program this way:

```
inp = input('Enter Fahrenheit Temperature: ')
try:
    fahr = float(inp)
    cel = (fahr - 32.0) * 5.0 / 9.0
    print(cel)
except:
    print('Error . . .  Please enter a number!')
```

```
python fahren2.py
Enter Fahrenheit Temperature: 72
22.22222222222222

python fahren2.py
Enter Fahrenheit Temperature: Sampath
Error . . . Please enter a number!
```

***Exercise 9:*** Re-write the program in Exercise 7 so that it prints an error message when the user enters invalid data – i.e. something other than a number. Save the new programme as ex9.py. The program should run as shown:

```
python ex9.py
Enter first number : 23
Enter second number : 1
23.0 is greater than 1.0


python ex9.py
Enter first number : 1
Enter second number : elijah@
Error . . . Please enter a valid number !
```

## 8. Short-circuit evaluation of logical expressions and guardian pattern

As with mathematical calculations, logical expressions are also evaluated left to right in Python.

In the expression x >= 2 and (x / y) > 2, x >= 2 gets evaluated first. Suppose x is less than 2, the first condition returns False. This makes the whole expression False regardless of whether the second condition (x/y) > 2 is True or False.

When Python detects that there is nothing to be gained by evaluating the rest of a logical expression it simply stops evaluating the expression. This is called short-circuiting.

***Exercise 10:*** Try the following sets of Python statements in the Python interpreter. Explain why the first set returns an answer while the second one terminates with a traceback.

```
>>> x = 1
>>> y = 0
>>> x >= 2 and (x/y) > 2

>>> x = 6
>>> y = 0
>>> x >= 2 and (x/y) > 2
```

***Exercise 11:*** Re-write the logical expression x >= 2 and (x/y) > 2 such that dividing by zero does not happen. You need to insert an additional Boolean evaluation into the expression.

## Review Questions

1. Write a pay computation programme that pays the worker 1.5 times the hourly rate for hours worked above 40 hours.

   Enter Hours: 45
   Enter Rate: 10
   Pay: 475.0

2. Re-write the program using the “try / except” structure to avoid tracebacks or unexpected errors occurring from invalid user input.

   Enter Hours: 20
   Enter Rate: nine
   Error, please enter numeric input

   Enter Hours: forty
   Error, please enter numeric input

3. Write a program that asks the user for their test score and prints out the grade. Aside from avoiding text input from the user, the program should also exit with an error message if the test score is not valid – i.e. less than 0 or greater than 100.

| Score | Grade |
|---|---|
| >= 90 | A |
| >= 80 | B |
| >= 70 | C |
| >= 60 | D |
| < 60 | Fail |

   Enter score: 95
   A

   Enter score: perfect
   Bad score

   Enter score: 106
   Bad score

   Enter score: 75
   C