CMP4*266*
Lab Session 2: Type conversions and Branching and Logical Expressions using Python

1. Objectives

a. Experiment with some type conversions involving strings, floats and integers.
b. Use of branching alternative paths of code execution using if-elif-else structure.
c. Experiment by testing logical and Boolean expressions using the Python command line
d. Integrate knowledge gained into simple date-processing software developed by student.

Note: Although most of the instructions in this sheet are written for command line interpreter, you are advised to save your python codes as scripts. To do that, go to your Z: drive and under the folder CMP4266 that you created in Lab-1 session create a folder called Lab2. You should save all your today's Python scripts under this Z:\ CMP4266 \Lab2 folder.

Simple types and conversions between types

In last lab session, 3 simple types of data (i.e. integer, float and string) were introduced but it was not discussed how data can be converted from one type to another type. Data type conversion becomes necessary at different stages of Python programming. For example Python provides built-in function *input* to read input form keyboard. However, *input* function only returns the data in string type, even you enter a number using the keyboard. To overcome problems like this Python offers different built-in functions to convert data types. More specifically in order to input whole numbers (integers) or numbers with fractions (floats) using the *input()* function we need to convert strings input from the keyboard by using *int()* and *float()* functions.

We might assign an *int* entered interactively by the user to i as follows:

```
i=int(input("enter a whole number "))
print(i)
```

Output

```
enter a whole number 12
12
```

Alternatively, we could do this in 2 steps as follows:

```
s=input("enter a whole number ")
i=int(s)
print(i)
```

Output

```
enter a whole number 12
12
```

Sometimes we will have to break complicated tasks up into smaller steps for readability and to easily debug the programme. We could have looked at the value of s in the above example, before trying to convert it to an integer.

1.1 A quick review of the 3 simple types of data which we used

a. Integers are whole numbers and are internally stored as a binary sign and a binary value. Integer arithmetic between 2 integers always has whole number results. 10//3 is 3, 2*4 is 8. In Python, as with arithmetic on paper, a fixed number of digits can be written and more space on the left can be added if large numbers exceed the space initially allocated. In most other programming languages, if allocated space is exceeded by a large number, binary integer storage will overflow.

b. Floats are numbers with fractions. Float arithmetic always has float results, so 10/3 is 3.33333333 and 2.0*4.0 is 8.0. Python floats are internally stored as a binary sign, a binary exponent and a binary mantissa typically using 64 bits of storage. (for a full explanation, see [1]) . If we use arithmetic between both integer and float data types we get a float result, so 2*3.0 is the same as 2.0*3 and both result in 6.0.

c. Strings are sequences of characters in Python, e.g. 'hello world' or "hello world".

1.2 Converting numbers as strings into integers and floats

If we try to carry out arithmetic directly on a decimal number stored as a string in Python, ,e.g. '10.5' * 4, this won't work, and we have to convert the string first to a number, e.g. making use of a function. float('10.5') gives us 10.5 , and float('10.5') * 4 is 42.0 .

If we want to convert a string to an integer, we can use the int() function instead, e.g. int('10'). But if our string contains a decimal point this will fail. But we can still do this job in 2 stages, e.g.

```
int('10.9')
```

Output

```
Traceback (most recent call last):

  File "<ipython-input-9-696bd5e02918>", line 1, in <module>
```

[1] http://steve.hollasch.net/cgindex/coding/ieeefloat.html

```
int('10.9')
```

ValueError: invalid literal for int() with base 10:

```
f = float('10.9')
int(f)
```

10

1.3 Converting between floats and ints

We can use the **float()** and **int()** functions on data of the other *type* to convert it. We could also convert any int into a float by multiplying it by 1.0 e.g.

```
int(9.6)
```
Output
9

```
float(8)
```
Output
8.0

```
8*1.0
```
Output
8.0

Use of the float() function is preferred, because then the source code expresses more clearly what the programmer *intended*.

1.4 Converting numbers into string format

There are 2 approaches, direct conversion, or interpolation of numbers into strings. We can use the str() function to convert a number into string directly.

```
str(8.5)
```
Output
'8.5'

```
str(3)
```
Output
'3'

We can also interpolate numbers into strings. Python uses %d and %f for ints and floats.

```
 s = "an int %d number" % 10
print(s)
```
Output
an int 10 number

```
s = "a float %f number" % 3.3
print(s)
```
Output
```
a float 3.300000 number
```

%f gives 6 decimal places. We can choose how many decimals we want. Formatting money amounts in a report we will typically want 2 decimal places.

```
"a float %.2f number" % 3.3
```
Output
```
'a float 3.30 number'
```

A number after the % and before the dot (.) gives us a field width e.g.:

```
"a float %10.2f" % 123456789.01
```
Output
```
'a float 123456789.01'
```

```
"a float %14.2f" % 123456789.01
```
Output
```
'a float 123456789.01'
```

Note how the too small width specification (10) was overridden, while the larger than enough one (14) was honoured.

If we want to interpolate more than one number into a string, we can use a *tuple*, which is a comma separated set of values in round brackets, to supply the data.

```
a=10
b=3.6
print("a float %f and an int %d" % (b, a))
```
Output
```
a float 3.600000 and an int 10
```

We could have used literals as previously or variables as here, or a mixture, but the left to right order of the values in the tuple has to match the order of the %d and %f conversion specifiers in the string.

1.5 Student exercises

a. Try out all of the above examples on the Python command line. Wherever you see literal values e.g. 10, 5.5 or "hello" try changing use of these expressions to use variables, after assigning literals to your variables e.g.

```
i=10
f=3.5
s="hello"
```

and then using i, f or s instead of 10,3.5 or "hello".

b. Try using %s to interpolate a string inside a string.

c. Experiment with getting %d, %f and %s int, float and string interpolations the wrong way around compared to the literal or variable values. E.G. see what happens when you evaluate:

   "Incorrect type %.2d" % 5.5

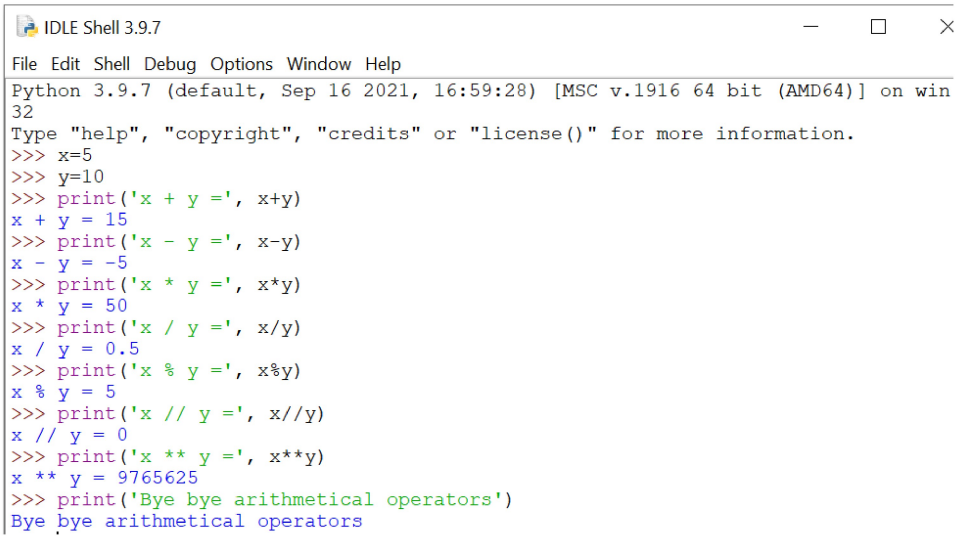
d. Try a selection of mismatched conversions and observe the results.

e. Experiment with putting a number between % and d, f or s when interpolating a value into a string e.g. %10s . Then try putting a dot before the number e.g. %.10s . Then try putting numbers both sides of the dot. e.g. %10.2f .

2. Operators

Operators are special symbols, which are used to carry out some kind of assignment or computations. These operators work on some values, which are termed operands. For example, in the expression: 2+4, + is an operator, and 2,4 are operands. Python supports many other operators which can be classified as follows:

- Arithmetic operators



- Assignment operators

```
>>> x=9# assigning 9 to variable x using '='
>>> x+=3 #equivalent to x=x+3
>>> print(x)
12
>>> x-=3#equivalent to x=x-3
>>> print(x)
9
>>> x/=2 #equivalent to x=x/2
>>> print(x)
4.5
>>> x%=3 #equivalent to x=x%3
>>> print(x)
1.5
>>> x**=2 #equivalent to x=x**2
>>> print(x)
2.25
>>> x//=2 #equivalent to x=x//2
>>> print(x)
1.0
>>> print('Bye bye assignment operators')
Bye bye assignment operators
>>>
```

- Comparison operators

```
>>> a=10
>>> b=5
>>> print(" The Output of 10 > 5 is : ", a>b )
 The Output of 10 > 5 is :  True
>>> print(" The Output of 10 < 5 is : ", a<b )
 The Output of 10 < 5 is :  False
>>> print(" The Output of 10 <= 5 is : ", a <= b )
 The Output of 10 <= 5 is :  False
>>> print(" The Output of 10 >= 5 is : ", a >= b )
 The Output of 10 >= 5 is :  True
>>> print(" The Output of 10 Equal to 5 is : ", a == b )
 The Output of 10 Equal to 5 is :  False
>>> print(" The Output of 10 Not Equal To 5 is : ", a != b )
 The Output of 10 Not Equal To 5 is :  True
```

- Logical operators

```
>>> a=False
>>> b=True
>>> print('a and b is', a and b)
a and b is False
>>> print('a or b is', a or b)
a or b is True
>>> print('not a is', not a)
not a is True
```

2.1 Student exercises

2.1.1 Construct the variable trace and predict the output.

```
amount1 = 10.0
amount2 = 40.0
amount1 *= 0.8
amount2 *= 0.9
tax = amount1 * 0.3 + amount2 * 0.2
total = amount1 + amount2 + tax
print(f"Total Cost: ${total}.")
```

amount1	amount2	tax	total

2.1.2. What is the print out of the following? Check your answer in Python.

a) print(3*'Yes',2*'!')
b) print(3*'Yes'+ 2*'!')
c) print (2**2**3)
d) print (2**(2**3))
e) print(10 // 3 ** 2 + (2 % 3))

2.1.3 Evaluate the following expressions and note the data type of the result:

Expression	Evaluates To	Data Type
3*2		
'3' + '2'		
'3' * 2		
3 / 2		
3 / /2		
3 + 2.0		
3 > 2		
3 != 2		
4 == 2**2		

3 > 2 and 10 > 11		
3 > 2 and not (10 > 11)		
format(5.1, '.2f')		
not (3>2 and 5 < 4)		

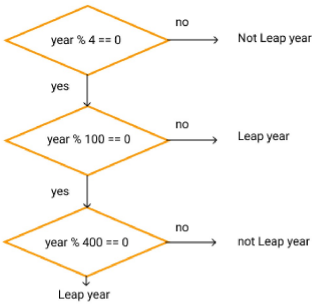
Errors

a) What is a syntax error? Give an example of one in Python.
b) What is a run-time error? Give an example of one in Python.
c) What is a logic error? Give an example of one in Python.

3. Branching alternate paths of code execution using if, elif and else

3.1 Write a Python program to determine whether a given year is a leap year based on the given workflow for calculating a leap year.

a) using Nested If
b) using Elif
c) using Conditional Statement

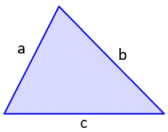


Add suitably indented print statements giving more information about each branch condition into your leap.py file e.g., one of these statements might be as follows:
print("year %d is divisible by 4 but not by 100" %y)

3.2 Write a python function to check whether three given numbers can form the sides of a triangle. If the numbers can form the side of the tringle calculate the perimeter and the surface using Heron's Formula.

Heron's Formula

We can use Heron's Formula to determine the area of a triangle when given the lengths of the sides.



Let $s = \frac{a+b+c}{2}$

Area $= \sqrt{s(s-a)(s-b)(s-c)}$

Hint: Three numbers can be the sides of a triangle if none of the numbers are greater than or equal to the sum of the other two numbers.

4. Moodle submission

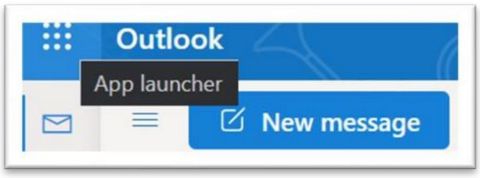
At the end of this lab session, zip the python files that you created in Sections 3 (3.1 and 3.2) and then upload the zip file in the Moodle. **Note:** It is important to complete the weekly labs in particular labs 1, 2, 3, 4, 5 and 6 because it contains questions that are part of the coursework.

Failure to complete and submit the answers to these labs on the Moodle before the deadline coding challenge may mean giving you **zero** on for this assessment component.

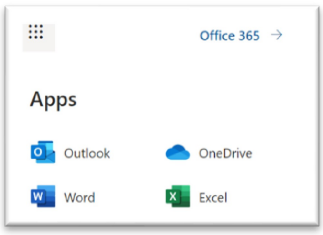
The zip file should be named using the following format StudetName_studentID.zip For example: OgertaElezaj_123456789.zip

Time to back up your works

- Go to your university email and select the App launcher from top right.



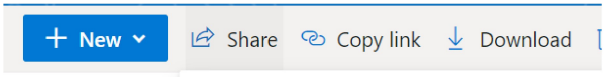
- Select OneDrive.



- Create a folder **CMP4266** and then create a subfolder **Lab 2**.
- Upload the script created in **task 3** to folder **Lab 2**.
- Keep the page open in the browser and ask the tutor to check it.

Peer review!

- Select the script uploaded in OneDrive in previous task.
- Click the "Copy link" button from the top menu to get a share link (or use "Share" directly).



- Email this link to the student next to you. Ask him/her to download and run your script.
- Verify he/she gets the same output.

5. Additional exercises

5.1 Find the minimum of three given numbers.
5.2 Test if a 5-digit number is a palindrome. (A palindromic is a number (such as 16461) that remains the same when its digits are reversed.
5.3 Check if a given number is of one digited or two digited or three digited or more than three digited.
5.4 Write a program that print three given numbers sorted from smallest to largest.
5.5 Write a program to determine whether an employee is owed any overtime. You should ask the user how many hours the employee worked in a specific week, as well as the hourly wage for this employee. If the employee worked more than 40 hours, you should print a message which says the employee is due some additional pay, as well as the amount due. The additional amount owed is 20% of the employees' hourly wage for each hour worked over the 40 hours.