New York University
School of Continuing and Professional Studies
Division of Programs in Information Technology

Introduction to Python

Exercise Solutions, Session 1

Ex 1.1 Assign two new integer objects and one float object to three variable names. Sum up the three variables and assign the resulting object to a new variable name. Indicate (by printing) the value and type of this resulting object.

```
#!/usr/bin/env python3

a = 5               # simple assignments
b = 10
c = 3.0

d = a + b + c       # simple summing using the numeric types

print(d)
print(type(d))      # type() returns a type object, this is printed
```

This demonstrates that an integer and float together in a math expression result in a float. This is a "behavior" that should be internalized. This will help in the "tip calculator" when calculating math expressions where a float value is expected.

Ex 1.2 Assign two new integer objects to two variable names. Multiply the two variables and assign the resulting object to a new variable name. Indicate (by printing) the value and type of this resulting variable's object.

```
#!/usr/bin/env python3

aa = 5          # simple assignments
bb = 10

cc = aa * bb    # simple numeric multiplication

print(cc)
print(type(cc)) # type() returns a type object, this is printed
```

Demonstrates that two integers in a math expression result in an integer. Again, this "behavior" should be internalized.

Ex 1.3 Starting with the sample code (make sure it is included exactly as written) sum up the values to produce the integer value **15**. (Hint: apply a conversion function to each string so it can be used as a number.)

```
#!/usr/bin/env python3

var = '5'               # note that these are strings, not integers
var2 = '10'
```

```
ivar = int(var)
ivar2 = int(var2)

var3 = ivar + ivar2     # we are using the return value of int()
                        # in a math expression

print(var3)
```

Numeric values often come into our program as strings (for example, text files or **input()**). In order to use them as numbers, we must convert. Use **int()** to produce a new integer value based on the string object, or **float()** to produce a float. **int()** will be essential in the homework assignments, since both will rely on **input()** to take numeric values from the user.

Ex 1.4 Take user input for an integer and print that value doubled.

```
#!/usr/bin/env python3

xx = input('Please enter an integer:  ')  # input() returns a str

yy = int(xx)
zz = yy * 2       # we must convert str to int with int()
                  # also, we can use the return value of int()
                  # in a math expression

print(xx + ' doubled is ' + str(zz) + '.')# we use + to concatenate strs
                                          # return val of str() can be
                                          # used directly in + expression
```

This exercise takes the previous exercise a step further by having the string numeric value come from **input()**. Another technique demonstrated here is the use of **str()** passed to **print()** that concatenates strings -- this is a very basic way to produce a single string made up of string and non-string (i.e. the integer in **yy**) objects.

Ex 1.5 Take user input for a 'place' and then greet the place enthusiastically.

```
#!/usr/bin/env python3

pp = input('Please enter a place name:  ')

print('Hello, ' + pp + '!')
```

This also shows how a string stored in a variable **pp** can be used in a string passed to **print()** by concatenating the strings.

Ex 1.6 Take user input for an integer and apply that many apostrophes to the phrase, "Hello, world!" (Hint: use the "string repetition" operator)

```
#!/usr/bin/env python

aa = input('Please enter an integer:  ')   # input() returns a str
```

```
bb = int(aa)
cc = '!' * bb              # str * int repeats the str that many times
                           # and returns a new str

print('Hello, World' + cc) # string concatenation
```

This demonstrates the use of the string repetition operator (*, which repeats a string when used with a string and int operand), which will be useful in the "Exponientation with tidy border" assignment.

Ex 1.7 Assign the float value 35.30 to a variable, then round the value to **35**.

```
#!/usr/bin/env python3

var = 35.30                # a float value

var2 = round(var)          # round takes a float and returns a float
                           # with only one argument, rounds to the
                           # nearest whole number (but still returns
                           # a float)

print(var2)
```

Rounding will be useful in the "tip calculator". With only one float argument, **round()** returns a float with no remainder, in this case **35**. The second argument (used in the next exercise) will allow for a rounding to a set number of decimal places.

Ex 1.8 Assign the float value **35.359958** to a variable, then round the value to two decimal places.

```
#!/usr/bin/env python3

var = 35.359958        # this many-places value is the kind of number
                       # one might expect from float division

var2 = round(var, 2)   # round the float value to 2 places (another float)

print(var2)
```

This use of **round()** will be useful in the "tip calculator" assignment, because we will want to round a float calculation to its nearest dollars and cents value (i.e., 2 decimal places).

Ex 1.9 Starting with the following code (make sure it is included *exactly as written*), divide the first number by the second number.

```
#!/usr/bin/env python3

var = "5"
var2 = "4"
```

```
var3 = int(5) / float(var2)  # return values from int() and float() are
                             # used in a math expression

print(var3)
```

This exercise combines our technique of converting string values to numeric types, and our use of a float value in any division from which we expect a float result. Note how we are using **int()** and **float()** in a mathmatical expression. We could have assigned **int()** and **float()** to individual variables, but instead we chose to simply put them into a math expression.