BYU IDAHO CSE 110 | Programming Building Blocks

03 Prepare: Preparation Material

Overview

In the previous lesson, you learned how to store words in variables and display them later in the program. In this lesson, you will learn how to work with numbers as well, including computing mathematical formulas and other types of expressions, or combinations of variables and operations.

For many types of programmers, working through calculations and manipulating numeric data is the primary reason they write programs. For example, scientists today use computers to simulate complex interactions that would be too difficult to work out by hand.

Preparation Material

I. DATA TYPES

To this point, you have been working with a type of data called *strings*, which can be sequences of any kind of characters (letters, numbers, symbols, spaces, etc.), such as **`"The book of: 1 Nephi"`**, **`"yellow puppy"`**, or **`"byui-cse.github.io/cse110-course"`**.

A sequence of numbers, such as **`"48212"`**, can be converted to a numeric data type and stored internally as a *number*; otherwise, the computer doesn't distinguish numbers from a sequence of letters. Once you have converted the string to a number, you can use it in mathematical formulas or expressions.

Please watch the following videos for more information:

» Numeric Data Types (6 mins)

» Demo: Numbers (6 mins)

II. USING DIFFERENT DATA TYPES

As shown above, when data is stored in a variable, the computer treats it as a certain data type (such as a string or an integer). You have to be conscious of these different data types when you want to mix them together, such as when you want to display a number alongside the text of a string.

The easiest way to display numbers alongside text, is to use a format string:

```
number = 7
print(f"The number is {number}.")
```

This is simple and works well, and is the preferred way to display a number in your print statement alongside text. But in order to properly use different data types, you'll need a little deeper understanding of the them as well as how operators like the plus sign work with them.

III. OPERATORS AND DIFFERENT DATA TYPES

Consider the plus operator `+`. When you add two strings, it glues them together, one right after the other (the technical term for this is that it *concatenates* them):

```
color = 'blue'
animal = 'horse'

# You can add, or concatenate, two strings together with +:
print(color + animal)
# This displays: bluehorse

# You can add many strings together, whether the strings are variables or directly in
# the quotation marks:
print(color + ' ' + animal + '!')
```

```
# This displays: blue horse!

# You can also save the result into a new string variable:
combined_words = color + ' ' + animal + '!'
print(combined_words)
```

On the other hand, if the data type of the variables is integer or floating point, the `+` operator actually performs the mathematical addition:

```
boys_count = 10
girls_count = 12

# Add two numbers together using the + operator:
print(boys_count + girls_count)
# This displays: 22

# You can save the result in a new variable to use later:
total_count = boys_count + girls_count
print(total_count)
```

IV. MIXING STRINGS AND NUMBERS

A potential problem arises if you try to mix strings and numeric data types together. For example, trying to add a string to an integer produces an error:

```
apple_count = 5

# Error on this line... You can't add strings and integers together!
print("You have " + apple_count + " apples") # ERROR!
```

The computer doesn't know how to add strings to numbers, which causes an error. In this case, what you want is for the computer to treat the number stored in the `apple_count` variable as a string instead of a number. It might seem obvious to you that this is the desired behavior, but the computer doesn't try to guess what you want, you must explicitly tell it when you want it to convert one type of data to another.

Converting data from one type to another is called type-casting, or "casting." That's just a fancy way of saying "treat it like a number please." To convert data to an integer, you put "int" before it, and then wrap it in parentheses, like this: **`int(your_data_here)`** . The same is true for converting it to a floating point number with the term **`float()`** or to a string with **`str()`** and so forth.

For example:

```
# This creates a new integer variable with the value of 10
# There is nothing magical about the "_num" in the variable name, it will just
# help us keep track of it
length_num = 50
width_num = 10

# If you add the numbers together, you would get the result you expect:
print(length_num + width_num) # This displays: 60

# You can convert the values to the strings "50" and "10" like this:
length_string = str(length_num)
width_string = str(width_num)

# The computer now thinks of these variable as two characters, the digit 5 followed
# by the digit 0, and the digit 1 followed by the digit 0.

# If you try to add the two strings together, it will concatenate them, or display
# one after the other:
print(length_string + width_string) # This displays: 5010
```

Whenever you get input from the user using **`input`**, it will be a string by default:

```
quantity_from_user = input("How many items do you have? ")

# The variable quantity_from_user is a string.
# To convert it to an integer you use the int(...) notation:
quantity_number = int(quantity_from_user)

# Because the quantity_number variable is an integer you can do math with it:
double_number = quantity_number * 2

# If you want to use these values in strings, you CANNOT just add them, you first
# have to convert them to a string:
```

```
# WRONG:
print("Twice as many is: " + double_number)

# Right:
double_string = str(double_number)
print("Twice as many is: " + double_string)

# You can also do this in one step:
# Right:
print("Twice as many is " + str(double_number))
```

Similar to the last example, you can combine the **`input`** and **`int`** commands into one line:

```
# Using two lines:
people_string = input("How many people are in the room? ")
people_number = int(people_string)

# Using one line:
people_number = int(input("How many people are in the room? "))

# The same works for floating point numbers:
length_number = float(input("What is the length? "))
```

Hint from Instructor:

In the examples on this page, we used variables like **`quantity_string`** and **`quantity_number`** to try to illustrate the data types of each variable, but when you write your code, you don't need to include the **`_string`** or **`_number`** parts. You should use variable names that are short and to the point such as: **`quantity`**, **`apples`**, **`length`**, etc.

V. EXPRESSIONS AND MATHEMATICAL OPERATIONS

Once you have numeric values stored in variables, you can perform a variety of complex mathematical operations. The following are common mathematical operators in Python:

Operator	Symbol	Example	Result
Add	`+`	`3 + 4`	`7`

Operator	Symbol	Example	Result
Subtract	`-`	`3 - 4`	`-1`
Multiply	`*`	`3 * 4`	`12`
Divide	`/`	`15 / 4`	`3.75`
Divide and drop remainder	`//`	`15 // 4`	`3`
Remainder or Modulus (Get the remainder that would result from dividing the first number by the second one.)	`%`	`25 % 7`	`4`
Exponent (To the power of)	`**`	`3 ** 4`	`81`

These operators follow standard mathematical orders of operation (where `*` happens before `+`), but you can force it to evaluate using parentheses. For example, `(3 + 4) * 2` will perform the addition first, and then multiple the result by 2.