In order to get started learning any programming language there are a number of concepts and ideas that are necessary. The goal of this module is to introduce you to the basic vocabulary of programming and some of the fundamental building blocks of Python. Please view the above video on these concepts.

Values and Data Types

A **value** is one of the fundamental things — like a word or a number — that a program manipulates. The values we have seen so far are `5` (the result when we added `2 + 3`), and `"Hello, World!"`. We often refer to these values as **objects** and we will use the words value and object interchangeably.

Note

Actually, the 2 and the 3 that are part of the addition above are values(objects) as well.

These objects are classified into different **classes**, or **data types**: `4` is an *integer*, and `"Hello, World!"` is a *string*, so-called because it contains a string or sequence of letters. You (and the interpreter) can identify strings because they are enclosed in quotation marks.

If you are not sure what class a value falls into, Python has a function called **type** which can tell you.

```
print(type("Hello, World!"))
print(type(17))
print("Hello, World")

```

ActiveCode: 1 (ch02_1)

Run

```
<type 'str'>
<type 'int'>
Hello, World
```

Not surprisingly, strings belong to the class **str** and integers belong to the class **int**.

Note

When we show the value of a string using the `print` function, such as in the third example above, the quotes are no longer present. The value of the string is the sequence of characters inside the quotes. The quotes are only necessary to help Python know what the value is.

In the Python shell, it is not necessary to use the `print` function to see the values shown above. The shell evaluates the Python function and automatically prints the result. For example, consider the shell session shown below. When we ask the shell to evaluate `type("Hello, World!")`, it responds with the appropriate answer and then goes on to display the prompt for the next use.

```
Python 3.1.2 (r312:79360M, Mar 24 2010, 01:33:18)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> type("Hello, World!")
<class 'str'>
>>> type(17)
<class 'int'>
>>> "Hello, World"
'Hello, World'
>>>
```

Note that in the last example, we simply ask the shell to evaluate the string “Hello, World”. The result is as you might expect, the string itself.

Continuing with our discussion of data types, numbers with a decimal point belong to a class called **float**, because these numbers are represented in a format called *floating-point*. At this stage, you can treat the words *class* and *type* interchangeably. We’ll come back to a deeper understanding of what a class is in later chapters.

```
1 print(type(3.2))
2
```

ActiveCode: 2 (ch02_2)

Run

```
<type 'float'>
```

What about values like `"17"` and `"3.2"` ? They look like numbers, but they are in quotation marks like strings.

```
1 print(type("17"))
2 print(type("3.2"))
3
```

ActiveCode: 3 (ch02_3)

Run

```
<type 'str'>
<type 'str'>
```

They’re strings!

Strings in Python can be enclosed in either single quotes (`'`) or double quotes (`"`), or three of each (`'''` or `"""`)

```
1 print(type('This is a string.') )
2 print(type("And so is this.") )
3 print(type("""and this.""") )
4 print(type('''and even this...''') )
5
```

ActiveCode: 4 (ch02_4)

Run

```
<type 'str'>
<type 'str'>
<type 'str'>
<type 'str'>
```

Double quoted strings can contain single quotes inside them, as in `"Bruce's beard"`, and single quoted strings can have double quotes inside them, as in `'The knights who say "Ni!"'`. Strings enclosed with three occurrences of either quote symbol are called triple quoted strings. They can contain either single or double quotes:

```
print('''"Oh no", she exclaimed, "Ben's bike is broken!"''')

```

ActiveCode: 5 (ch02_5)

Run

```
"Oh no", she exclaimed, "Ben's bike is broken!"
```

Triple quoted strings can even span multiple lines:

```
message = """This message will
span several
lines."""
print(message)

print("""This message will span
several lines
of the text.""")

```

ActiveCode: 6 (ch02_6)

Run

```
This message will
span several
lines.
This message will span
several lines
of the text.
```

Python doesn't care whether you use single or double quotes or the three-of-a-kind quotes to surround your strings. Once it has parsed the text of your program or command, the way it stores the value is identical in all cases, and the surrounding quotes are not part of the value.

```
1 print('This is a string.')
2 print("""And so is this.""")
3
```

ActiveCode: 7 (ch02_7)

Run

```
This is a string.
And so is this.
```

So the Python language designers usually chose to surround their strings by single quotes. What do think would happen if the string already contained single quotes?

When you type a large integer, you might be tempted to use commas between groups of three digits, as in `42,000`. This is not a legal integer in Python, but it does mean something else, which is legal:

```
1 print(42000)
2 print(42,000)
3
```

ActiveCode: 8 (ch02_8)

Run

```
42000
42 0
```

Well, that's not what we expected at all! Because of the comma, Python chose to treat this as a *pair* of values. In fact, the print function can print any number of values as long as you separate them by commas. Notice that the values are separated by spaces when they are displayed.

```
print(42, 17, 56, 34, 11, 4.35, 32)
print(3.4, "hello", 45)

```

ActiveCode: 9 (ch02_8a)

Run

```
42 17 56 34 11 4.35 32
3.4 hello 45
```

Remember not to put commas or spaces in your integers, no matter how big they are. Also revisit what we said in the previous chapter: formal languages are strict, the notation is concise, and even the smallest change might mean something quite different from what you intended.

Check your understanding

sdat-1: How can you determine the type of a variable?

- a) Print out the value and determine the data type based on the value printed.
- b) Use the type function.
- c) Use it in a known equation and print the result.
- d) Look at the declaration of the variable.

Check Me | Compare Me

Correct!! The type function will tell you the class the value belongs to.

sdat-2: What is the data type of 'this is what kind of data'?

- a) Character
- b) Integer
- c) Float
- d) String

Check Me | Compare Me

Correct!! Strings can be enclosed in single quotes.

Type conversion functions

Sometimes it is necessary to convert values from one type to another. Python provides a few simple functions that will allow us to do that. The functions `int`, `float` and `str` will (attempt to) convert their arguments into types int, float and str respectively. We call these **type conversion** functions.

The `int` function can take a floating point number or a string, and turn it into an int. For floating point numbers, it *discards* the decimal portion of the number - a process we call *truncation towards zero* on the number line. Let us see this in action:

```
print(3.14, int(3.14))
print(3.9999, int(3.9999))       # This doesn't round to the closest int!
print(3.0,int(3.0))
print(-3.999,int(-3.999))        # Note that the result is closer to zero

print("2345",int("2345"))        # parse a string to produce an int
print(17,int(17))                # int even works on integers
print(int("23"))

```

ActiveCode: 10 (ch02_20)

Run

```
3.14 3
3.9999 3
3 3
-3.999 -3
2345 2345
17 17
23
```

The last case shows that a string has to be a syntactically legal number, otherwise you'll get one of those pesky runtime errors. Modify the example by deleting the `bottles` and rerun the program. You should see the integer `23`.

The type converter `float` can turn an integer, a float, or a syntactically legal string into a float.

```
print(float("123.45"))
print(type(float("123.45")))

```

ActiveCode: 11 (ch02_21)

Run

```
123.45
<type 'float'>
```

The type converter `str` turns its argument into a string. Remember that when we print a string, the quotes are removed. However, if we print the type, we can see that it is definitely str.

```
1 print(str(17))
2 print(str(123.45))
3 print(type(str(123.45)))
4
```

ActiveCode: 12 (ch02_22)

Run

```
17
123.45
<type 'str'>
```

Check your understanding

sdat-3: What value is printed when the following statement executes?

```
print( int(53.785) )
```

- a) Nothing is printed. It generates a runtime error.
- b) 53
- c) 54
- d) 53.785

Check Me | Compare Me

Correct!! The int function truncates all values after the decimal and prints the integer value.