

Beginner May 22 - Python Notes

Variables

A name (*variable or constant*) is not itself typed, and can be attached or re-attached to different objects over its lifetime. For extended naming conventions and advice, see [PEP 8](#).

```
>>> my_first_variable = 1
>>> my_first_variable = "Last one, I promise"
>>> print(my_first_variable)
...
"Last one, I promise"
```

[Comments](#) in Python start with a `#` that is not part of a string, and ends at line termination. Unlike many other programming languages, Python does not support multi-line comment marks. Each line of a comment block must start with the `#` character.

Comments are ignored by the interpreter:

```
# This is a single line comment.

x = "foo" # This is an in-line comment.

# This is a multi-line
# comment block over multiple lines --
# These should be used sparingly.
```

Introduction

Numbers

There are three different kinds of built-in numbers in Python : `ints`, `floats`, and `complex`. However, in this exercise you'll be dealing only with `ints` and `floats`.

ints

`ints` are whole numbers. e.g. 1234, -10, 20201278.

Integers in Python have [arbitrary precision](#) -- the amount of digits is limited only by the available memory of the host system.

floats

`floats` are numbers containing a decimal point. e.g. 0.0, 3.14, -9.01.

Floating point numbers are usually implemented in Python using a `double` in C (*15 decimal places of precision*), but will vary in representation based on the host system and other implementation details. This can create some surprises when working with floats, but is "good enough" for most situations.

You can see more details and discussions in the following resources:

- [Python numeric type documentation](#)
- [The Python Tutorial](#)
- [Documentation for `int\(\)` built in](#)
- [Documentation for `float\(\)` built in](#)
- 0.30000000000000004.com

Arithmetic

Python fully supports arithmetic between `ints` and `floats`. It will convert narrower numbers to match their less narrow counterparts when used with the binary arithmetic operators (+, -, *, /, //, and %). When division with /, // returns the quotient and % returns the remainder.

Python considers `ints` narrower than `floats`. So, using a float in an expression ensures the result will be a float too. However, when doing division, the result will always be a float, even if only integers are used.

```
# The int is widened to a float here, and a float type is returned.  
>>> 3 + 4.0  
7.0
```

```
>>> 3 * 4.0
12.0
>>> 3 - 2.0
1.0
# Division always returns a float.
>>> 6 / 2
3.0
>>> 7 / 4
1.75
# Calculating remainders.
>>> 7 % 4
3
>>> 2 % 4
2
>>> 12.75 % 3
0.75
```

If an int result is needed, you can use `//` to truncate the result.

```
>>> 6 // 2
3
>>> 7 // 4
1
```

To convert a float to an integer, you can use `int()`. Also, to convert an integer to a float, you can use `float()`.

```
>>> int(6 / 2)
3
>>> float(1 + 2)
3.0
```

Boolean

Python represents true and false values with the `bool` type. There are only two values in this type: `True` and `False`. These values can be bound to a variable:

```
>>> true_variable = True
>>> false_variable = False
```

We can evaluate Boolean expressions using the `and`, `or`, and `not` operators:

```
>>> true_variable = True and True
>>> false_variable = True and False

>>> true_variable = False or True
>>> false_variable = False or False

>>> true_variable = not False
>>> false_variable = not True
```

Python programming language is a great language to learn whether you are a beginner learner or someone who has experience in multiple languages.

Key Vocabulary

- **Syntax:** a set of rules that defines the structure of a language. This includes symbols and characters that you need to use in a specific order.
- **Data types:**
 - `int`: The **integer** data type represents both positive and negative whole numbers.
 - `str`: The **String** data type represents a set of characters.
 - `bool`: The **boolean** data type has only two possible values and are used to determine the logical truth value of a statement, either `True` or `False`.
- **Variables:** used to **store** information of *any* data type. Variables are given names to allow programmers to easily access data stored and manipulate or change the value being stored.

Why Python?


- Python is easy to learn!
 - Python has fewer syntax rules than many other programming languages.
 - It still follows a very similar structure and basis of other languages (i.e defining variables, functions, classes, similar data type operations & rules, similar data structures)
- Python is fast to program in
 - Because of its pseudocode-like nature, it is easy to translate your plan to code fast.
 - This means you can speed up the time it takes for you to answer questions in an interview setting.
- Python is a powerful language
 - Python is an **open-source language**. This means that many programmers across the world have contributed to the development of this language and continue to add more features and packages. This makes the language highly adaptable and easy to use.
 - Programmers around the world use it to code in web development, data science, machine learning, robotics, and so much more!
 - There are currently more than 200,000 Python packages in the world and counting!

How does a programming language communicate with a computer?

You may be asking the following questions and we are here to answer them! Computers at its core can only understand a series of 0s and 1s. As you may imagine, this makes it difficult for humans to communicate with computers. No need to worry! This is where programming languages like Python come in. Our programming language ***translates*** English-like code into the 0s and 1s that the computer understands. There are many layers into how a programming language does that we won't get into.

It is important that when writing in Python we follow the rules, or **syntax**, of the language.

Syntax is a set of rules that defines the structure of a language. This includes symbols and characters that you need to use in a specific order.

 **Real World Example:** Think about if you were trying to learn how to speak a new language like Spanish, French, or German. You would have to learn the *syntax* of that language including how to pronounce words, punctuation, grammar, and spelling. Learning a new programming language is very similar in that there are *syntax* rules to follow to learn how to properly "speak" the language with your computer.

Data Types

Information or data can have many representations: numbers, letters, characters, a list or sequence, etc. We group these larger buckets of data into types.

Data Type	Definition	Syntax	Examples
<code>int</code>	The integer data type represents both positive and negative whole numbers.	Integers are numbers without any decimals or commas. The <code>-</code> symbol may be used to identify negative numbers	<code>1, -5</code>

<code>str</code>	The String data type represents a set of characters. This includes letter characters (a-z, A-Z), number characters (0-9), and special characters (!, #, \$, etc).	To define a String, or a set of characters, you need to use a pair of quotation marks. They can be either single or double quotation marks.	<code>"hello", 'Hello!', "2"</code>
<code>bool</code>	The boolean data type has only two possible values and are used to determine the logical truth value of a statement, either <code>True</code> or <code>False</code>	Booleans have only two values and must be written as <code>True</code> or <code>False</code>	<code>True, False</code>

Note: Python supports *many* different types of data types but in this lesson we will only cover the 3 common data types: integers, strings, and booleans. To explore more data types, check out the [full list of built-in types in Python](#).

Variables

Variables allow users to **store** information of *any* data type. Programmers are able to easily access information stored in a variable and even manipulate or change the value being stored.

Let's take a look at the syntax to define a variable.

Variable Syntax

```
varName = value
```

- `varName`: variable name that references the data being stored. You can give a variable any name but it must be String, or word, with no spaces. Some programmers use **camel case** (i.e `varName`) or **snake case** (i.e `var_name`) to separate variable names with multiple words.
- `=`: The equal sign is used to assign or reassign a value to the variable name.

- `value`: Placeholder for the value of your data type that you are storing.

Note: In Python you do not need to specify the data type being stored in a variable. This is different than how variables are defined in some other programming languages like Java, JavaScript, or C++.

Receiving input and sending output through the console

Thus far we learned about how to create data and store them in Python but what do we do with all of this information? In this section we will learn about how to receive information from the user using the `input` function and how to output or send information back to the user using the `print` function.

Python has a few built in **functions**, or prewritten blocks of code. This makes it easier for programmers like us to get our code to do something without having to reinvent the wheel. We will talk more about functions later and how they are created. For now you should just know that functions hold lines of code and can be easily accessed, or called, by using its name followed by parenthesis.

`print` function

The `print` function allows us to send a message to the screen, or **console**. The function takes in any object and then converts it into a string to output to the screen. Let's take a look at the syntax to use the `print` function.


Print Function Syntax

```
print(object1, object2)
```

- `print`: Keyword to call the set of code that sends a message to the screen
- `()`: Parentheses are used to indicate we want to call, or let the computer know to run the set of code associated with a function.
- `object1|object2`: reference to any object or data you want printed. This can be a string, variable that stores data, etc. They will then be converted into a string format to appear on the screen.
- `,`: This *optional* character is used to separate objects if you want to print out multiple objects.

Now it's your turn!

Try practicing using the `print` function to print out different types of data types.

 **Tip:** Don't forget to follow **syntax** rules. Remember to check spelling, capitalization, and punctuations are all correct!

`input` function

Now we know how to print a message to the screen, but how do we **receive** information from the user? The `input` function allows coders to ask users a prompt and then **records** their response.

It's not enough to just use the `input` function to ask the user something, we need a way to **store** their response and use it in our code. For this we need to also assign the response from the `input` function to a **variable**. Let's take a look at the syntax for using the `input` function

Input Function Syntax

```
answer = input(prompt)
```

- `answer`: This is the variable name we use to store the response from the user after they answer the prompt. Remember variable names can be anything, it does not have to be named `answer`.
- `=`: equal sign is used to assign the response to the variable name.
- `input`: Keyword to call the set of code that sends a prompt question to the screen and records the user's response.
- `()`: Parentheses are used to indicate we want to call, or let the computer know to run the set of code associated with a function.
- `prompt`: A string value representing the message to be sent to the user. Don't forget that since this is a string you will need to use a pair of either single (' ') or double quotation marks " " .