Data Types in Python

* Why is Python Useful for Data Science
  + It’s open source: You are not locked into using a particular company’s products
  + It has a large community: You can easily find answers to questions
  + It’s interpreted: You can run code piece by piece without creating an executable file
  + There is an abundance of 3rd party packages available: Prewritten Python packages can be used for data science, video games, website back-ends, computer vision, and more
  + It offers Easy C integration: Speed-critical routines can be written in the low-level language C and easily integrated
* When would you not use Python?
  + When high performance or low memory usage is important.
  + When direct access to hardware is required
  + When OS specific GUI features are desired
  + When doing embedded systems programing, where speed and memory footprint matter
* How a Computer Program Works
  + Computer programs take three general steps:
    - **Loading data from the outside world into memory.** This involves loading data into variables, which are typically stored in your main memory (RAM). This data could come from a file, database queries, API calls, keystrokes, mouse clicks, etc.
    - **Manipulating the data in memory.** Once data are stored in variables, your Python script can easily and efficiently access them.
    - **Export the data back to the outside world.** Once your script does some processing, it must export the variables and store the result. This could mean writing to a file, writing to a database, or lighting pixels on the screen.
    - Most programming textbooks exclusively (and appropriately) focus on the second step. However, many students get lost when making a "real" program from what they learn in books. Typically, they miss how to load data into variables or how to export those variables back to disk or screen!
* Running Python
  + Method 1: Interactively from the command line (python or ipython). This method allows the user to type and evaluate one line of code at a time. It is called a REPL, for Read, Evaluate, Print, Loop. It reads the user's input, evaluates it, prints the output, then loops back to prompting the user for input. This is useful for quickly experimenting with short code snippets.
    - Try it out!
      * Throughout these lessons, we'll ask you to experiment with Python. You can do so by:
      * Using Repl.it: Navigate to https://repl.it/languages/python3, where you can easily type code, run it, and see output in a user-friendly environment.
      * Or...
      * In the terminal: Open your terminal (hit command+space, then type terminal). Next, type ipython at the command prompt. Now, you can evaluate individual lines of code such as 1 + 1 by typing them and pressing enter.
  + Method 2: As a script (python <script\_name>). By saving Python code in a file, we can run it automatically. This is useful when writing complex programs, or automated programs that you want to run repeatedly. We can also store frequently used routines in a file and import them in an interactive session.
  + Method 3: From a Jupyter Notebook (jupyter notebook). Jupyter is also a REPL, but it allows us to easily save our inputs and outputs in a user-friendly way.
  + Pro tip: Consider using the program ipython ("Interactive Python"). This program was created as an enhancement for Python's bare-bones REPL. It adds useful features such as syntax highlighting and "magic commands" that begin with %. Jupyter Notebook is actually built on top of IPython — it passes each input cell to ipython in the background and shows you the result!
* Python Enhancement Protocols
  + Python enhancement protocols, or PEPs, are documents that explain the why behind each new Python feature. A new PEP must be written for the community to consider a new feature.
  + Here are some example PEPs. No need to read them all now, but keep them in mind and refer to them as you develop your Python coding ability.
  + [PEP 8 — Style Guide for Python Code](https://www.python.org/dev/peps/pep-0008/). Ever wonder how many spaces to use, the maximum line length, or when to uppercase variables? Read this!
  + [PEP 20 — The Zen of Python](https://www.python.org/dev/peps/pep-0020/). Every language has an underlying philosophy. The gist of Python's is: Prefer readable slow code over hard-to-read performant code.
  + [PEP 465 — A Dedicated Infix Operator for Matrix Multiplication](https://www.python.org/dev/peps/pep-0465/). This more recent PEP explains the rationale behind adding a new Python 3 operator for matrix multiplication.
* Operators
  + In Python, operators act on one or two objects and evaluate to a single object. For example:
    - 1 + 1 evaluates to 2. (The numbers here are the objects, and the + is the operator.)
    - 'a' + 'b' evaluates to 'ab'. (The letters here are the objects, and the + is again the operator.)
  + An integer added to an integer evaluates to an integer. A string added to a string evaluates to a string.
    - Make sure you actively try these yourself in the REPL. Pro tip: Behind the scenes, Python actually converts each operator into a function call! For example, 'a' + 'b' is evaluated as 'a'.\_\_add\_\_('b'). The dot indicates that we are calling a method of the 'a' object. We’ll look at objects in more detail in Lesson 2.
  + It's important to know the data type on each side of the operator. This defines what the operator will produce. For example, think about what you might expect the following operators to evaluate to, then try them out:
    - 'a' \* 3
    - 3 \* 'a'
    - 'a' \* 2.5
  + The results of these operators do "make sense," given what we know about multiplication. People created many of Python’s libraries to follow the same logic. So, we can usually guess correctly what might work.
  + Note that there is no "correct" answer with regard to these outputs. How these operators act on specific data types was determined by a person. They could have just as easily been defined so that multiplying a string by an integer resulted in a syntax error. (In fact, this is the case in many languages, such as C.)
* The Assignment Operator
  + The assignment operator is a single equals sign (=). It associates a name with a value. For example:
    - num\_cookies = 3
    - first\_name = 'Sally'
  + The assignment operator is different from the equals sign we're familiar with from math. Instead of asserting that the two sides are equal, the assignment operator defines a name. It first evaluates the right-hand side, then associates the name on the left-hand side to the result of the evaluation. The two lines of code above work like so:
    - An integer object is created with a value of 3. Then, we point the name num\_cookies to it.
    - A string object is created with a value of 'Sally'. Then, we point the name first\_name to it.
  + Here is an example of where newbies often get confused:
    - num\_cookies = 3
    - num\_cookies = num\_cookies + 1
    - Clearly, num\_cookies cannot be equal to one more than itself! However, recall that = is not a comparison. Instead, it's a definition. So, we first evaluate the right-hand side, num\_cookies + 1, to be 4. Next, we point the name num\_cookies to it.
* The Comparison Operator
  + In Python, the equality sign you know from math class is represented as double equals ==. This operator compares two objects and evaluates to either True, if the values are the same, or False, if the values differ. For example:
    - 2 == 2 → True
    - 2 == 3 → False
    - 'hi' == 'hi' → True
    - 'hi' == 'bye' → False
  + It is important to note that == is an operator just like +. Compare the following:
    - num\_cookies = 1 + 2 → 3
    - is\_confirmed = 'yes'
    - is\_confirmed == 'yes' → 'True'
  + Pro tip: Use parentheses to avoid any possibility of ambiguity when combining multiple operators. For example, ('1' + '2') == '12'.
  + Python has a number of comparison operators. You can view them, and much more, in the online documentation.
* Data Types and Binary
  + Your computer stores everything (integers and text characters) in binary as 1s and 0s. Data types tell the computer how to interpret these 1s and 0s. For example, the sequence 01000001 can represent either the integer 65 or the character 'A'.
* Python Data Types
  + Python comes with only a few data types. The complete list of built-in data types is:
    - None.
    - Booleans.
    - Numbers (i.e., int, float, long, complex).
    - Strings.
    - Lists.
    - Tuples.
    - Sets.
    - Dictionaries.
  + Data science libraries function to add new data types to Python — ndarray (NumPy), Series (Pandas), and DataFrame (Pandas).
* Integers
  + Whole numbers are integers with an optional + or - prefix — e.g., 3, 82, 38218, +3, -71.
  + In Python, integers can be as large as desired. For example, 27381732198731297381273127.
  + But be careful! In Python 2, an integer divided by an integer is an integer. This way, all operations maintain the same data type. However, this is a common source of confusion for new users. For example, 2 / 3 = 0 and 5 / 2 = 2. (It truncates the result, i.e., the (floored) quotient.)
  + Pro tip: The % modulus operator returns the remainder.
  + In Python 3, the divisions above produce "floating point" (non-integer) numbers, e.g., 2 / 3 = 0.6666666... and 5 / 2 = 2.5. (Python 3 additionally adds the quotient operator // that will you give the same integer result as / in Python 2.)
  + We will be working in Python 3, but it's important that you understand the differences in case you encounter code written in Python 2.
  + Try experimenting with the built-in function type() (e.g., type(72)).
* Boolean
  + Boolean values are either True or False. Internally, they are stored as an integer which is 1 if True and 0 if False
  + Boolean variables are often called **flags**, as they indicate whether or not something is present. For example, email\_is\_valid could be a flag indicating a valid email address.
    - When naming a Boolean variable, it is often helpful to prefix it with is\_ or has\_ to clearly indicate that it is a flag.
      * Example: has\_header, is\_first, is\_blue.
  + Recall that the comparison operator == always evaluates to True or False.
* Strings
  + A string is a sequence of characters. To indicate a string in Python code, enclose it with quotation marks.
  + Single quotes (') and double quotes (") are equivalent in Python. If your string contains a single quote, then enclose it with double quotes, and vice versa. Examples include:
    - first\_name = 'Charles'
    - html = '<a href="https://generalassemb.ly/"></a>'
  + Try type('some text').
* Escape Characters
  + Strings may also contain characters with special meanings. For example, a newline (\n), a tab (\t), or a special Unicode character.
  + To use these, prefix them with a backslash, as shown above. You can enter a backslash as \\. For example:
    - 'Line One\nLine Two'
    - "A list:\n\t- Bullet 1\n\t- Bullet 2"
  + To view these lines as intended without the backslashes, just print them (e.g., print("A list:\n\t- Bullet 1\n\t- Bullet 2")).
* Multi-Line Strings
  + A third way of defining strings exists: the multi-line string. By enclosing text in three quotation marks (""" or '''), you can write multi-line strings that preserve newlines. For example, this code...
  + """
  + Header
  + \* Line Two
  + \* Line Three
  + """
  + ...becomes "\nHeader\n\t- Line Two\n\t- Line Three".
* Data Type Conversions
  + Most built-in data types have a built-in function (a **constructor**) that assists with conversions. For example, we saw that float(3) creates a new float object. str(72) returns a string representation of the integer 72. bool(82) returns True for any non-zero integer. None, True, and False do not have constructors, as they are singular values.
* Indexing
  + Because a string is a sequence of characters, we can get the value of any individual character. In Python, we accomplish this using zero-based indexing, where the first character is referenced by index 0. So, the last character in the string would be referenced by one less than the number of characters.
  + We can use the built-in len() function on any Python object that might have a length. Here, the length of a string is the number of characters it contains.
  + We can use the indexing operator [] to get a particular character in the string.
  + Suppose we define title = 'President'. Then, title[0] is the character 'P'. len(title) evaluates to 9, as there are nine characters in 'President'. So, title[8] is the character t because of zero-based indexing. More generically, title[len(title)-1] evaluates to 't'.
  + Because title is a name that just points to the 'President' object, you can also index into a newly created string! For example, 'giraffe'[0] evaluates to 'g'. Here, a string object is created and then indexed into. When we wrote title[0], it looked up the object that title pointed to and then indexed into it.
  + Later on, we'll discuss **slicing**, an extension of indexing.
  + Explore: What do you think might happen if you index by a negative number? What if you index by a number greater than the length of the string? Do your answers agree with the choices the Python language designers made?
* Everything Is an Object!
  + With Python, you'll hear that everything is an object. But what does this mean? An object is just a particular grouping of functions and variables. An object's functions are called methods — functions that "belong to" an object (and typically act on its variables).
  + For example, a string contains variables — characters. It also contains methods that act on the string, for example, upper() converts the string to uppercase. These methods are called by adding a period after the variable name.
  + Let’s look at 'cat'.upper(). This first creates a new string object and then calls its upper() method. This function returns an object; in this case, the uppercase string 'CAT'. If you set animal = 'cat', then you can call upper() in the same way with animal.upper(). Here, it looks up the object animal points to, then calls its upper() method.
  + To view the methods available in a REPL or Jupyter, type 'cat'. (with a trailing period), then hit the tab key (only if you are using ipython). Alternatively, you can call the built-in dir() function (short for "directory") — e.g., dir('cat'). To see what each method does, call the help() function (e.g., help('cat'.upper)). You can also refer to the [documentation online](https://docs.python.org/3/library/stdtypes.html#string-methods). It’s worth it for practice to try out each of the string methods, as you'll be using them often.
  + Try out some of the other string methods, too. Define a new string variable name containing your full name. Then, using the techniques above, try out some of the available methods. In particular, try name.capitalize(), name.title(), name.islower(), name.replace('a', 'b'), name.center(40), and name.center(40, '\*').
  + Pro tip: Note that 'cat'.upper is a name, too. It points to an object — a function object. Try type('cat'.upper). When Python sees an open parenthesis immediately following a name, it knows that you must be making a function call.