Lesson Design Notes: Python Intro for Libraries

Lesson Description

This lesson introduces programming in Python for library and information workers with little or no previous programming experience. It uses examples that are relevant to a range of library use cases and is designed as a prerequisite for other Python lessons that will be developed in the future (e.g., web scraping, APIs). The lesson uses the JupyterLab computing environment and Python 3.

Target Audience

The target audience for this lesson includes library and information professionals who are new to programming and are interested in learning Python to support various library-related tasks. This lesson is especially useful for those who plan to engage in data manipulation, web scraping, or working with APIs in the future.

Prerequisites

Learners should have the following skills/knowledge before starting this lesson:

1. Understanding of files and directories, including what a working directory is.
2. Installation of Python and JupyterLab.
3. Downloading and setting up the dataset to be used in the lesson before the workshop begins.

Lesson Learning Objectives

After following this lesson, learners will be able to:

- Navigate the JupyterLab interface and run Python cells within a notebook.
- Assign values to variables, identify data types, and display values in a Jupyter Notebook.
- Create and manipulate lists in Python, including indexing, slicing, appending, and removing items to manage data collections effectively.
- Call and nest built-in Python functions, and use the help function to understand their usage and troubleshoot errors.
- Use Python libraries like Pandas to import modules, load tabular data from CSV files, and perform basic data analysis.
- Apply for loops to iterate over collections, using the accumulator pattern to aggregate values and trace variable states to predict loop outcomes.
- Manipulate pandas DataFrames to select data, calculate summary statistics, sort data, and save results in various formats, demonstrating basic data handling and analysis proficiency.
- Write Python programs using conditional logic with if, elif, and else statements, including Boolean expressions and compound conditions within loops.
- Construct Python functions that encapsulate tasks, manage parameters, local, and global variables, and return values to enhance code modularity and readability.
- Transform complex datasets into a tidy format using pandas functions like `melt()` for reshaping, `group by ()` for aggregation, and `to_datetime()` for date handling. Address practical challenges and demonstrate the benefits of tidy data for analysis.
- Create and customize data visualizations using Pandas and Plotly, generating various plot types (line, area, bar, histogram) to analyze trends and draw insights from time-series data.
- Prepare for advanced Python topics such as web scraping and APIs.

Questions

The following questions are designed to prime the learner for the content in this lesson:

1. What are the basic data types in Python, and how can they be used in library-related tasks?
2. How can Python scripts automate routine tasks in a library setting?
3. What are the benefits of using JupyterLab for running Python code?

Setup Instructions:

Installing Python Using Anaconda

Python is a popular language for research computing, and great for general-purpose programming as well. Installing all of its research packages individually can be a bit difficult, so we recommend Anaconda, an all-in-one installer.

Regardless of how you choose to install it, please make sure you install Python 3.6 or above. The latest 3.x version recommended on Python.org is fine.

We will teach Python using JupyterLab, a programming environment that runs in a web browser (JupyterLab will be installed by Anaconda). For this to work you will need a reasonably up-to-date browser. The current versions of the Chrome, Safari and Firefox browsers are all supported (some older browsers, including Internet Explorer version 9 and below, are not).

Windows - Video tutorial

1. Open anaconda.com/download with your web browser.
2. Download the Anaconda for Windows installer with Python 3. (If you are not sure which version to choose, you probably want the 64-bit Graphical Installer *Anaconda3-…-Windows-x86_64.exe*)
3. Install Python 3 by running the Anaconda Installer, using all of the defaults for installation except make sure to check *Add Anaconda to my PATH environment variable*.

macOS - Video tutorial

1. Open anaconda.com/download with your web browser.
2. Download the Anaconda Installer with Python 3 for macOS (you can either use the Graphical or the Command Line Installer).
3. Install Python 3 by running the Anaconda Installer using all of the defaults for installation.

Linux

Note that the following installation steps require you to work from the shell. If you aren't comfortable doing the installation yourself stop here and request help from the workshop organizers.

1. Open anaconda.com/download with your web browser.
2. Download the Anaconda Installer with Python 3 for Linux.

3. Open a terminal window and navigate to the directory where the executable is downloaded (e.g., `cd ~/Downloads`).
4. Type `bash Anaconda3` and press Tab to auto-complete the full file name. The name of file you just downloaded should appear.
5. Press Enter (or Return depending on your keyboard). You will follow the text-only prompts. To move through the text, press Spacebar. Type `yes` and press Enter to approve the license. Press Enter (or Return) to approve the default location for the files. Type `yes` and press Enter (or Return) to prepend Anaconda to your PATH (this makes the Anaconda distribution the default Python).
6. Close the terminal window.

JupyterLab

We will teach Python using JupyterLab, a part of a family of Jupyter tools that includes Jupyter Notebook and JupyterLab, both of which provide interactive web environments where you can write and run Python code. If you installed Anaconda, JupyterLab is installed on your system. If you did not install Anaconda, you can install JupyterLab on its own using conda, pip, or other popular package managers.

Download the data

---

1. Download this zip file and save it to your Desktop.
2. Unzip the `data.zip` file, which should create a new folder called `data`.
3. Create a new folder on your Desktop called `lc-python` and put the `data` folder in this folder.

This lesson uses circulation data in multiple CSV files from the Chicago Public Library system. The data was compiled from records shared by the Chicago Public Library in the data.gov catalog. Please do not download the circulation data from data.gov since the dataset you downloaded following the steps above has been altered for our purposes.

Instructor Notes:

General Notes

It's all right not to get through the whole lesson.

This lesson is designed for people who have never programmed before, but any given class may include people with a wide range of prior experience. We have therefore included enough material to fill a full day if need be, but expect that many offerings will only get as far as the introduction to Pandas.

Don't tell people to Google things.

One of the goals of this lesson is to help novices build a workable mental model of how programming works. Until they have that model, they will not know what to search for or how to recognize a helpful answer. Telling them to Google can also give the impression that we think their problem is trivial. (That said, if learners have done enough programming before to be past these issues, having them search for solutions online can help them solidify their understanding.) It's also worth quoting Trevor King's comment about online search: "If you find anything, other folks were confused enough to bother with a blog or Stack Overflow post, so it's probably not trivial."

Episodes

Each lesson should have more than one episode. Each episode is a separate file in the Carpentries Workbench **For each episode**, provide the following information:

Episode 1: Getting Started

Objectives

- Identify applications of Python in library and information science environments by the end of this lesson.
- Launch JupyterLab and create a new Jupyter Notebook.
- Navigate the JupyterLab interface, including file browsing, cell creation, and cell execution, with confidence.
- Write and execute Python code in a Jupyter Notebook cell, observing the output and modifying code as needed.
- Save a Jupyter Notebook as an .ipynb file and verify the file's location in the directory within the session.

Lesson Content

Why Python?

Python is a popular programming language for tasks such as data collection, cleaning, and analysis. Python can help you to create reproducible workflows to accomplish repetitive tasks more efficiently.

This lesson works with a series of CSV files of circulation data from the Chicago Public Library system to demonstrate how to use Python to clean, analyze, and visualize usage data that spans over the course of multiple years.

Challenge: Python in Libraries

There are a lot of ways that library and information science folks use Python in their work. Go around the room and have helpers and co-teachers share how they have used Python.

Learners: Can you think of other ways to use Python in libraries? Do you have hopes for how you'd like to use Python in the future?

Solution: Python in Libraries

Here a few areas where you might apply Python in your work:

- **Metadata work:** Many cataloging teams use Python to migrate, transform, and enrich metadata that they receive from different sources. For example, the `pymarc` library is a popular Python package for working with MARC21 records.
- **Collection and citation analysis:** Python can automate workflows to analyze library collections. In cases where spreadsheets and OpenRefine are unable to support specific forms of analysis, Python is a more flexible and powerful tool.
- **Assessment:** Library workers often need to collect metrics or statistics on some aspect of their work. Python can be a valuable tool to collect, clean, analyze, and visualize that data in a consistent way over time.
- **Accessing data:** Researchers often use Python to collect data (including textual data) from websites and social media platforms. Academic librarians are often well-positioned to help teach these researchers how to use Python for web

scraping or querying Application Programming Interfaces (APIs) to access the data they need.

- **Analyzing data:** Python is widely used by scholars who are embarking on different forms of computational research (e.g., network analysis, natural language processing, machine learning). Library workers can leverage Python for their own research in these areas, but also take part in larger networks of academic support related to data science, computational social sciences, and/or digital humanities.

Callout: Alternatives to Juypter

There are other ways of editing, managing, and running Python code. Software developers often use an integrated development environment (IDE) like PyCharm, Spyder or Visual Studio Code (VS Code), to create and edit Python scripts. Others use text editors like Vim or Emacs to hand-code Python. After editing and saving Python scripts you can execute those programs within an IDE or directly on the command line.

Episode 2: Variables and Types

Objectives

- Write Python to assign values to variables.
- Print outputs to a Jupyter notebook.
- Use indexing to manipulate string elements.
- View and convert the data types of Python objects.

Key Questions

- How can I store data in Python?
- What are some types of data that I can work with in Python?

Lesson Content

Use variables to store values.

Variables are names given to certain values. In Python, the `=` symbol assigns a value to a variable. Here, Python assigns the number `42` to the variable `age` and the name `Ahmed` in single quotes to a variable `name`.

```
age = 42

name = 'Ahmed'
```

Callout: Naming Variables

Variable names:

- Cannot start with a digit.
- Cannot contain spaces, quotation marks, or other punctuation.
- *May* contain an underscore (typically used to separate words in long variable names).
- Are case-sensitive. `name` and `Name` would be different variables.

Use `print()` to display values.

You can print Python objects to the Jupyter notebook output using the built-in function, `print()`. Inside the parentheses, you can add the objects that you want to print, which are known as the `print()` function's arguments.

```
print(name, age)

Ahmed 42
```

In Jupyter notebooks, you can leave out the `print()` function for objects, such as variables, that are on the last line of a cell. If the final line of a Jupyter cell includes the name of a variable, its value will display in the notebook when you run the cell.

```
name
age

42
```

Format output with f-strings

F-strings provide a concise and readable way to format strings by embedding Python expressions within them. You can format variables as text strings in your output using an f-string. To do so, start a string with `f` before the open single (or double) quote. Then add any replacement fields, such as variable names, between curly braces `{}`.

```
f'{name} is {age} years old'

'Ahmed is 42 years old'
```

Variables must be created before they are used.

If a variable doesn't exist yet, or if the name has been misspelled, Python reports an error called a `NameError`.

```
print(eye_color)
NameError: name 'eye_color' is not defined
```

The last line of an error message is usually the most informative. In this case, it tells us that the `eye_color` variable is not defined.

Variables can be used in calculations.

We can use variables in calculations as if they were values. We assigned `42` to `age` a few lines ago, so we can reference that value within a new variable assignment.

```
age = age + 3

f'Age equals: {age}'

Age equals: 45
```

Every Python object has a type.

Everything in Python is some type of object, and every Python object will be of a specific type. Understanding an object's type will help you know what you can and can't do with that object.

You can use the built-in Python function `type()` to find out an object's type.

```
print(type(140.2),

      type(age),

      type(name),

      type(print))
```

<class 'float'> <class 'int'> <class 'str'> <class 'builtin_function_or_method'>

Types control what operations (or methods) can be performed on objects.

An object's type determines what the program can do with it.

```
5 - 3

2
```

We get an error if we try to subtract a letter from a string:

```
'hello' - 'h'

TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

Use an index to get a single character from a string.

We can reference the specific location of a character (individual letters, numbers, and so on) in a string by using its index position. In Python, each character in a string (first, second, etc.) is given a number, which is called an index. Indexes begin from `0` rather than `1`. We can use an index in square brackets to refer to the character at that position.

```
library = 'Alexandria'

library[0]

A
```

Use a slice to get multiple characters from a string.

A slice is a part of a string that we can reference using `[start:stop]`, where `start` is the index of the first character we want and `stop` is the last character. Referencing a string slice does not change the contents of the original string. Instead, the slice returns a copy of the part of the original string we want.

```
library[0:3]

Ale
```

Use the built-in function `len` to find the length of a string.

The `len()` function will tell us the length of an item. In the case of a string, it will tell us how many characters are in the string.

```
len('Babel')

5
```

Variables only change value when something is assigned to them.

Once a Python variable is assigned, it will not change value unless the code is run again. The value of `older_age` below does not get updated when we change the value of `age` to `50`, for example:

```
age = 42

older_age = age + 3

age = 50

f'Older age is {older_age} and age is {age}'

Older age is 45 and age is 50
```

Challenge: F-string Syntax

Use an f-string to construct output in Python by filling in the blanks with variables and f-string syntax to tell Christina how old she will be in 10 years.

Tip: You can combine variables and mathematical expressions in an f-string in the same way you can in variable assignment.

```
name = 'Christina'

age = 23

f'{____}, you will be ______ in 10 years.'
```

Solution

```
f'{name}, you will be {age + 10} in 10 years.'

'Christina, you will be 33 in 10 years.'
```

Challenge: Swapping Values

Draw a table showing the values of the variables in this program after each statement is executed. In simple terms, what do the last three lines of this program do?

```
x = 1.0

y = 3.0

swap = x

x = y

y = swap
```

Solution

```
swap = x  #  x = 1.0 y = 3.0 swap = 1.0

x = y     #  x = 3.0 y = 3.0 swap = 1.0

y = swap  #  x = 3.0 y = 1.0 swap = 1.0
```

These three lines exchange the values in `x` and `y` using the `swap` variable for temporary storage. This is a fairly common programming idiom.

Challenge: Predicting Values

What is the final value of `position` in the program below? (Try to predict the value without running the program, then check your prediction.)

```
initial = "left"

position = initial

initial = "right"
```

Solution

```
initial = "left"  # Initial is assigned the string "left"

position = initial  # Position is assigned the variable initial, currently
"left"

initial = "right"  # Initial is assigned the string "right"
```

```
print(position)

left
```

The last assignment to position was "left."

Challenge: Can you slice integers?

If you assign `a = 123`, what happens if you try to get the second digit of `a`?

Solution

Numbers are not stored in the written representation, so they can't be treated like strings.

```
a = 123

print(a[1])

TypeError: 'int' object is not subscriptable
```

Challenge: Slicing

We know how to slice using an explicit start and end point:

```
library_name = 'Library of Babel'

f'library_name[1:3] is: {library_name[1:3]}'

'library_name[1:3] is: ib'
```

But we can also use implicit and negative index values when we define a slice. Try the following (replacing `low` and `high` with index positions of your choosing) to figure out how these different forms of slicing work:

1. What does `library_name[low:]` (without a value after the colon) do?
2. What does `library_name[:high]` (without a value before the colon) do?
3. What does `library_name[:]` (just a colon) do?

4. What does `library_name[number:negative-number]` do?

Solution

1. It will slice the string, starting at the

Component Elements

When using these, please clearly indicate what's part of the block. For their appearance, please look at the Workbench Component Guide. Use them in line with your lesson content.

- ☐ **Challenges/Exercises/Activities** Provide practical tasks for learners to complete.
- ☐ **Challenges with Solutions** Provide solutions for the challenges to facilitate discussions and deepen understanding.
- ☐ **Discussion** Facilitate discussions around the challenges to deepen understanding.
- ☐ **Callout Blocks** One of the key elements of our lessons is our callout blocks, which give learners and instructors a bold visual cue to stop and consider a caveat or exercise.
- ☐ **CHECKLIST Callouts** Created with the checklist tag, and are a way to emphasize particular steps more strongly.
- ☐ **TESTIMONIAL Callouts** Created with the testimonial tag and are quotations from previous learners or instructors.
- ☐ **SPOILER Callouts** Created with the spoiler tag and are a way to provide additional details/content that can be expanded and collapsed on demand.
