2.1 Programming with Python and Jupyter

Learning goals

- Compare advantages and disadvantages of Python for data science.
- Define programs, variables, and expressions.
- Define functions and components of functions.
- Use the Jupyter notebook interface.

Python

Python is a an open-source programming language for data science. ***Open source*** means a user can access the source code and create new functions and libraries to share with other users. Python is free to download and use, and several user interfaces are available to support reading, writing, and using Python for data science.

Python was developed during the late 1980s by Guido van Rossum and initially released in 1994. A new version of the language, Python 3.0, was introduced in 2008 and included major changes to the Python syntax to make code more readable. As a result, Python 3.0 is not ***backward compatible*** with earlier versions. Ex: Python 2.7 programs cannot run on Python 3.0 or later. Smaller releases, denoted by Python 3.x, make incremental changes to the language.

Table 2.1.1: Advantages and disadvantages of Python for data science.

Advantages	Disadvantages
Readability: Python functions from the same library, or collection of functions, using consistent syntax.	Consistency: Different libraries may have different syntax conventions.
Popularity: Python is popular in data science and elsewhere in industry, which means resources for learning Python are widely available.	Memory: Python uses more computer memory than other programming languages.
Innovation: New data science models and technologies are constantly added to Python.	Speed: Other programming languages such as Julia perform computations on datasets more quickly than Python.

PARTICIPATION ACTIVITY

2.1.1: Python background.

1) Python is a programming language primarily used by data scientists.
   - True
   - False

2) Python 2.7 programs cannot run on Python 3.7 interpreters.
   - True
   - False

3) Python code is more difficult to read than code in most other programming languages.
   - True
   - False

Programs, variables, and expressions

A computer ***program*** is a set of lines of code executing one at a time. Each ***line*** in a program contains a single instruction.

- A ***variable*** is a named item used to store a value.
- Variables are created by performing an ***assignment*** using the = symbol. Ex: A weather model predicts a high temperature of 90F. Thus, a value of 90 can be assigned to the variable `predicted_temp` using the code `predicted_temp=90`.
- ***Expressions*** are lines of code that return a specific value when evaluated. Expressions usually describe simple calculations. Ex: `temp_error = actual_temp - predicted_temp`.

The ***Python interpreter*** executes code written in the Python programming language. Python is a human-readable language, but not a machine-readable language. The Python interpreter's job is to translate written Python code into machine language.

Entering lines of code into an interpreter one by one is inefficient. An ***interactive development environments*** or ***IDE*** is a software application for writing, testing, and running code in a single window. In data science, using an IDE allows programmers to write code that can be updated and re-used or shared with other data scientists.

PARTICIPATION ACTIVITY 2.1.2: Python interpreter.

Python code

```
high_temp = 72

low_temp = 64

avg_temp = (high_temp + low_temp)/2

print('Average temperature:', avg_temp)
```

Python interpreter

Name	Value
high_temp	72
low_temp	64
avg_temp	68

Output

```
Average temperature:  68
```

Animation content:

Step 1: A box appears labeled Python code. Inside the box is a Python program.
high_temp = 72
low_temp = 64
avg_temp = (high_temp + low_temp)/2
print('Average temperature:', avg_temp)
Step 2: A box appears labeled Python interpreter. A table in the box has two columns: Name and Value. high_temp is moved from the Python code to the name column and 72 is moved from the Python code to the value column. low_temp is moved to the name column and 64 is moved to the value column.
Step 3: A new line of code appears in the Python box.
avg_temp = (high_temp + low_temp)/2
In the interpreter, avg_temp moves to the name column and 68 moves to the value column.
Step 4: A new line of code appears in the Python box.
print('Average temperature:', avg_temp)
The text "Average temperature" moves to a box labeled output, and 68 moves from the interpreter box to the output box.

Animation captions:

1. The Python program uses daily high and low temperatures to calculate a daily average temperature in San Diego, CA.
2. The Python interpreter reads and runs this code line by line. Values for the variables high_temp and low_temp are assigned by the interpreter.
3. (high_temp + low_temp)/2 is calculated as (72+64)/2, then the resulting value is assigned to the variable avg_temp.
4. The print() function combines the text in quotations with the value of avg_temp.

PARTICIPATION ACTIVITY 2.1.3: Python variables and expressions.

A data scientist wants to model daily pollution levels in a large city, measured as levels of fine particulate matter (`FPM`). Fine particulate matter levels depend on features like traffic level and temperature.

1) The model predicts a fine particulate matter value of 20 $\mathbf{mg/m^3}$. How would the predicted value be assigned to a variable, `pred_FPM`?

   - `pred_FPM = 20`
   - `pred_FPM: 20`
   - `pred_FPM == 20`

2) The code `pred_FPM*0.001` is an expression.

   - True
   - False

3) How are most Python programs in data science developed?

- Writing code directly in an interpreter
- Writing code in an interactive development environment
- Searching for code online

Python functions

Most data scientists use pre-defined functions to model and visualize datasets. A ***function*** is a command that performs computations.

- A ***parameter*** is an input to a function, like a dataset, feature, or function option. Some functions have default settings for parameters.
- An ***argument*** is a value provided to a function's parameter. Ex: In the function `predict(traffic='high')`, `traffic` is the parameter and `'high'` is the argument.
- A ***return*** is the function's output. Data science functions may return numerical values or tables, which can be assigned to a variable. Functions for visualization return plots instead of values or text. Some functions do not automatically display a return value.

Functions help data scientists write code that is more consistent and easier to read and understand. Data science models use complex mathematical formulas, often with multiple steps. A data scientist using functions can fit a model in one line of code vs. hundreds of lines.

PARTICIPATION ACTIVITY 2.1.4: A first program in Python.

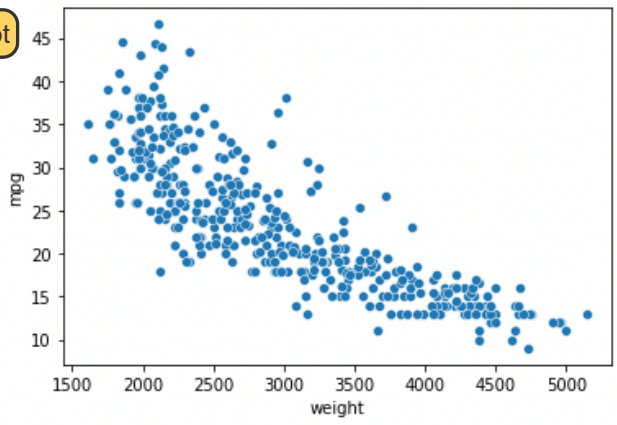
Dataset car_data

mpg	cylinders	displacement	horsepower	weight
18.0	8	307.0	130.0	3504
15.0	8	350.0	165.0	3693
18.0	8	318.0	150.0	3436
16.0	8	304.0	150.0	3433
17.0	8	302.0	140.0	3449

Code

```
scatterplot(data=car_data, x='weight', y='mpg')
```

Plot



Animation content:

Step 1: A dataset appears with five columns: mpg (miles per gallon), cylinders, displacement, horsepower, and weight. Step 2: The Python function scatterplot() appears. A large blank space is between the parentheses. Step 3: Three parameters appear in the blank space: data=, x=, and y=. Step 4: The column headers for weight and mpg are highlighted in the dataset. The highlights animate down to the Python function. The full function is scatterplot(data=car_data, x='weight', y='mpg'). Step 5: A scatter plot of all cars in the car_data dataset appears with weight on the horizontal axis and mpg on the vertical axis. There is a downward trend in the scatter plot: cars with higher weights have lower mpg.

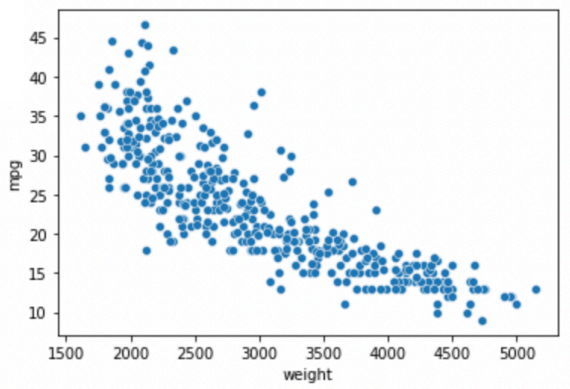
Animation captions:

1. A data scientist wants to plot the relationship between the weight of a car and the car's gas consumption, measured in miles per gallon (mpg).
2. The scatterplot() function can be used to make a plot with two features.
3. The scatterplot() function has three parameters: data, x, and y.
4. 'weight' and 'mpg' are assigned as arguments to x and y. The dataset, car_data, is assigned to the data parameter.
5. The scatterplot function returns a plot with weight on the x-axis and mpg on the y-axis.

PARTICIPATION ACTIVITY 2.1.5: Functions.

Match the parts of the Python code below.

```
scatterplot(data=car_data, x='weight', y='mpg')
```



If unable to drag and drop, refresh the page.

Return Function Parameters

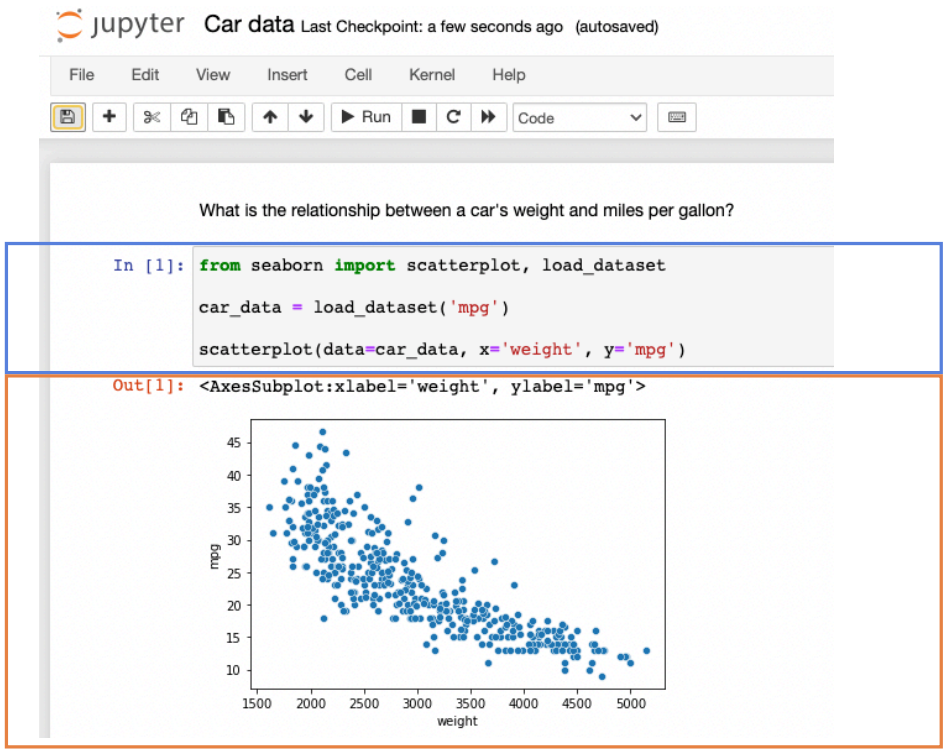
	`data=car_data, x='weight', y='mpg'`
	`scatterplot()`
	scatter plot

Reset

Jupyter

Jupyter is an interactive development environment (IDE) for writing and testing code in data science and scientific computing. In Jupyter, code is written in an interactive document called Jupyter Notebook. Notebooks contain code, outp and text all in one place, which makes Jupyter convenient for testing, presenting, and sharing code.

PARTICIPATION ACTIVITY 2.1.6: Jupyter notebooks.



Animation content:

Step 1: A screenshot of the Jupyter notebook interface appears. The notebook interface has a text cell, then a code cell. Step 2: The first code cell contains the Python program:
from seaborn import scatterplot, load_dataset
car_data = load_dataset('mpg')
scatterplot(data=car_data, x='weight', y='mpg')
Step 3: The "Run" button is highlighted on the menu bar.
Step 4: The scatter plot appears in an output cell labeled Out[1] below the code cell.

Animation captions:

1. Jupyter notebooks have code cells, text cells, and output cells. The notebook's first cell is text: "What is the relationship between a car's weight and miles per gallon?"
2. Code cells are numbered and labeled with "In". Text cells are not labeled. Ex: Code cell In[1] contains code to load the dataset and produce a scatter plot.
3. Clicking the "Run" button executes Python functions contained in a single code cell.
4. Running the code generates a scatter plot below the code cell. Output cells are labeled with "Out" and the matching code cell number. Ex: Out[1]

PARTICIPATION ACTIVITY 2.1.7: Jupyter notebook.

1) Small segments of code and output in Jupyter notebooks are called _____.
   - blocks
   - cells
   - chunks

2) The Run button executes all code contained in a Jupyter notebook.
   - True
   - False

3) Jupyter can only be used to write and execute code in Python.
   - True
   - False

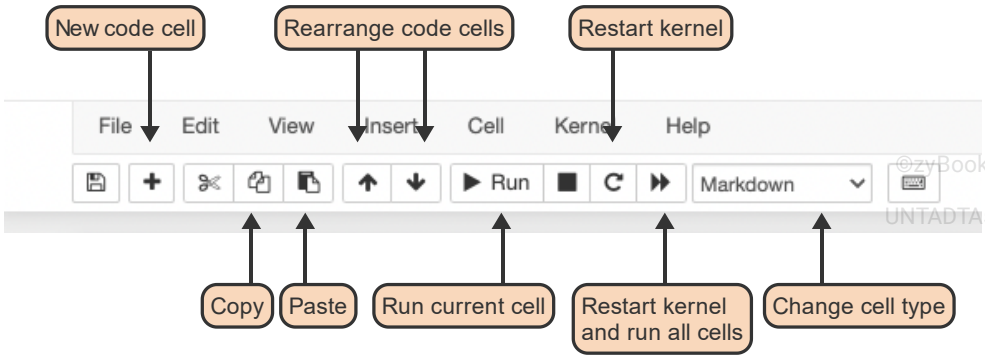
Using Jupyter notebooks

Jupyter notebooks are used throughout this zyBook to illustrate data science applications. Each sample Jupyter notebook contains all the code to run a short analysis of a dataset. Students are encouraged to modify the sample code to test new parameter values or different modeling functions.

Sample Jupyter notebooks will refresh each time the zyBook is loaded. Clicking the "File" menu and "Download as" saves a notebook into a local working directory. Jupyter notebooks can be downloaded as the actual notebook file, or as a stand-alone document with code and output.

For information about installing and using Jupyter Notebook on a personal computer, check out the Jupyter installation guide.

PARTICIPATION ACTIVITY 2.1.8: Jupyter menu icons.



Animation content:

Step 1: The Jupyter notebook menu bar appears. Step 2-7: Arrows appear pointing to each button on the menu bar.

Animation captions:

1. The Jupyter notebook menu bar has several useful shortcuts.
2. The New Cell button adds a new code cell to the existing notebook, directly below the active cell.
3. The Copy button makes a copy of the active code cell. The Paste button inserts the copied cell into the notebook.
4. The up and down arrows move the position of a code cell up or down in the Jupyter notebook.
5. The Restart button restarts the code kernel, or backend. The kernel is an interpreter that runs the Python code and checks for errors.
6. The Run button runs the current code cell. The Restart and Run All button restarts the kernel before running all code cells, in order.
7. The Cell Type drop-down changes each input cell from text (Markdown) or code (Code).

PARTICIPATION ACTIVITY 2.1.9: Jupyter notebooks in the zyBook.

1) Jupyter notebooks cannot be changed within the zyBook.
   - ○ True
   - ○ False

2) What happens to the Jupyter notebook when a zyBook section is refreshed?
   - ○ The Jupyter notebook will revert to the original code.
   - ○ The Jupyter notebook will display changes made by the student.
   - ○ The Jupyter notebook will display changes made by other students.

3) What does the double right triangle button do in a Jupyter notebook?
   - ○ Run current cell only
   - ○ Run all cells
   - ○ Restart the kernel and run all cells

4) What does the + button do in a Jupyter notebook?
   - ○ Add a new cell directly below the current cell
   - ○ Add a new cell at the end of the notebook
   - ○ Create a new notebook

Getting started with Jupyter notebooks.

Full screen

The Jupyter notebook loads the miles per gallon dataset and creates a scatter plot of miles per gallon against weight and engine size (number of cylinders) for each car. The notebook also calculates summary statistics for weight.

- Click the double right arrow icon to restart the kernel and run all cells.
- Examine the code and text below.
- Double click the text cell. Add a new heading and text inside the first text cell.
- Copy code cell `In[3]`. In the new cell, change the `'weight'` feature to `'horsepower'`.
- Click "File", then "Download as". Download the Jupyter notebook as a PDF or HTML file.

This is a heading

This is a text cell. Data scientists use text cells in a Jupyter notebook to write comments abo findings. Double click this Markdown cell to change the text, and click Run to format the text.

This is a smaller heading

Text cells can contain:

- Bulleted lists
- Like this

Or

1. Numbered lists
2. Like this

In [1]:
```
# This is a code cell
# In Python, hashtags are used to write comments, or text
# It's good practice to use comments to write notes above a function
```

In [2]:
```
# Import seaborn library
import seaborn as sns

# Load the miles per gallon dataset and assign the dataset to mpg
mpg = sns.load_dataset('mpg')
```


2.2 Python data types

Learning goals

- Define container types.
- Define indexing, ordering, and mutability.
- Define and compare list, tuple, set, and dictionary types.
- Construct and use container-type objects.

Containers

A ***container*** is a data type for which each value consists of multiple elements. An ***element*** is a value of any type. Python has four built-in containers: list, tuple, set, and dictionary types. Additional containers are available by importing Python modules.

Different elements of a container value may have different types. Ex: `['abc', 91, False]` is a list with string, integer, and boolean elements.

Elements of a container value may be ***ordered*** or ***indexed***. List and tuple elements are ordered. List, tuple, and dictionary elements are indexed. Set elements are neither ordered nor indexed. Ex:

- `['abc', 91, False]` and `[False, 91, 'abc']` are different lists, since the order is significant.
- If `list = ['abc', 91, False]`, then `list[0]` returns 'abc'.

A container is ***mutable*** if an individual element can be changed, or ***immutable*** if not. List, set, and dictionary are mutable, but tuple is immutable. Ex: `list[2] = 3.29` is valid, but `tuple[2] = 3.29` is invalid.

Table 2.2.1: Container types.

	Ordered	Indexed	Mutable
list	yes	yes	yes
tuple	yes	yes	no
set	no	no	yes
dict	no	yes	yes

PARTICIPATION ACTIVITY 2.2.1: Containers.

1) A sorted collection of strings can be stored as a list.
   - True
   - False

2) Ordered containers are also indexed.
   - True
   - False

3) Indexed containers are also ordered.
   - True
   - False

4) If a variable is assigned to an immutable value, the variable cannot subsequently be modified.
   - True
   - False

List type

A ***list*** is an ordered, indexed, and mutable container. A list is created by surrounding a sequence of variables or literals with brackets. Ex:

- `list = [10, 'abc']` creates a list with elements 10 and 'abc'.
- `list = []` creates an empty list.

A list element is accessed by appending an integer index, within brackets, to the variable name. The index begins at 0. Since a list is mutable, elements can be updated, added, or removed. Ex: `list[0] = 33` replaces the first element of `list` with 33.

Table 2.2.2: Example list functions, methods, and operators.

Example	Description
`len(list)`	Finds the length of the list
`list.append(x)`	Adds x to the end of the list
`list.pop(x)`	Removes the element with an index of x and returns that element
`list.remove(x)`	Removes x from the list
`list1 + list2`	Concatenates the two lists

Functions and methods

*A **method** is a function that is defined within a class and operates on class objects. A method call has an object prefix. Ex: `array.fill(1)` is a method and `sqrt(4)` is a function.*

Lists.

Full screen

- Click the double right arrow icon to restart the kernel and run all cells.
- Examine the code below.
- In cell [4], change the argument of the `pop()` method to `1`. Restart the kernel and run all cells to see the changes.
- Click "File", then "Download as". Download the Jupyter notebook as a PDF or HTML file.

In [1]:
```
# Create a list
list = [33, 2, 14]
```

In [2]:
```
# Find the length of the list
len(list)
```

Out[2]: 3

In [3]:
```
# Append 'apples' to the list
list.append('apples')
list
```

Out[3]: [33, 2, 14, 'apples']

In [4]:
```
# Remove and display the first list element
list.pop(0)
```

Out[4]: 33

In [5]:
```
# Print the list
list
# Current list contains 2, 14, and 'apples'
```


Out[5]: [2, 14, 'apples']

PARTICIPATION ACTIVITY 2.2.2: List modification.

Write a statement that performs the desired action. Assume the list `housePrices = ['$140,000', '$550,000', '$480,000']` exists.

1) Update the price of the second item in `housePrices` to '$175,000'.

Check Show answer

2) Add a price to the end of the list with a value of '$1,000,000'.

Check Show answer

3) Remove the first element from `housePrices`, using the `pop()` method.

Check Show answer

4) Remove '$140,000' from `housePrices`, using the `remove()` method.

Check Show answer

Tuple type

A ***tuple***, pronounced "tuhple" or "toople", is an immutable list. Like lists, tuples are ordered and indexed with integers. Unlike lists, individual elements of a tuple cannot be updated, added, or removed. The methods `append()`, `remove()`, and `pop()` do not work with tuples.

List functions and operators that do not change individual elements do work with tuples. Ex:

- `len(tuple)` returns the number of elements in `tuple`.
- `tuple1 + tuple2` returns a tuple consisting of `tuple1` followed by `tuple2`.

Tuples are used instead of lists when the elements never change. Ex: A tuple might store the latitude and longitude of geographic landmarks, since the location of a landmark is unlikely to change.

A ***named tuple*** is similar to a tuple. Like tuples, named tuples are ordered, indexed, and immutable. Unlike tuples, named tuple elements may be indexed with strings as well as integers. A named tuple type is constructed with ***namedtuple*** class of the Collections package, which must be imported to a Python program.

Tuples and named tuples.

Full screen

- Click the double right arrow icon to restart the kernel and run all cells.
- Examine the code below.
- In cell [4], uncomment `whiteHouse[1] = 50`. Restart the kernel and run all cells to see the changes.
- Click "File", then "Download as". Download the Jupyter notebook as a PDF or HTML file.

In [1]:
```
# Load namedtuple class
from collections import namedtuple
```

In [2]:
```
# Create a tuple
whiteHouse = (38.8977, 77.0366)
```

In [3]:
```
# Print tuple elements and length
print('Coordinates:', whiteHouse)
print('Latitude:', whiteHouse[0])
print('Longitude:', whiteHouse[1])
print('Tuple length:', len(whiteHouse))
```

```
Coordinates: (38.8977, 77.0366)
Latitude: 38.8977
Longitude: 77.0366
Tuple length: 2
```

In [4]:
```
# The commented line below produces an error
# whiteHouse[1] = 50
```

In [5]:
```
# Create a named tuple type Car and object chevyBlazer
Car = namedtuple('Car', ['make', 'model', 'price', 'horsepower', 'sea
chevyBlazer = Car('Chevrolet', 'Blazer', 32000, 275, 8)
chevyBlazer
```

PARTICIPATION ACTIVITY | 2.2.3: Tuples.

1) Create a new variable `point` that is a tuple containing the strings `'X string'` and `'Y string'`.

Check Show answer

2) If the value of variable `friends` is the tuple `('Cleopatra', 'Marc', 'Seneca')`, then what is the result of `len(friends)`?

Check Show answer

Set type

A ***set*** is an unordered, unindexed, mutable container. No elements in a set may share the same value. A set literal is written with curly braces, indicating that elements are not ordered, rather than square brackets. Ex:

- `set = {33, 4,'abc'}` creates a set of three elements.
- `{33, 4,'abc'}` and `{'abc', 4, 33}` are the same set, since sets are not ordered.

A string, list, or tuple can be converted to a set with the `set()` function. Duplicate elements are removed. Ex: `set([33, 4,'abc', 4])` returns `{33, 4,'abc'}`.

Since sets are not indexed, individual elements cannot be accessed with bracket notation. Since sets are mutable, elements can be added or removed.

Sets. Full screen

- Click the double right arrow icon to restart the kernel and run all cells.
- Examine the code below.
- Did any of the results give an empty set? Why?
- Click "File", then "Download as". Download the Jupyter notebook as a PDF or HTML file.

```
In [1]: # Create a set
        set = {'abc', 'xyz'}

In [2]: # Add 'banana' to the set
        set.add('banana')
        set

Out[2]: {'abc', 'banana', 'xyz'}

In [3]: # Remove 'abc' from the set
        set.remove('abc')
        set

Out[3]: {'banana', 'xyz'}

In [4]: # Remove a random element from the set
        set.pop()
        set

Out[4]: {'xyz'}

In [5]: # Display the union of set and {'xyz', 44}
        set.union({'xyz', 44})

Out[5]: {44, 'xyz'}
```


PARTICIPATION ACTIVITY 2.2.4: Basic sets.

1) What's the result of `set(['A', 'Z'])`?

- A set that contains `'A'` and `'Z'`.
- A list with the following elements: `['A', 'Z']`.
- Error: invalid syntax.

2) What's the result of `set(10, 20, 25)`?

- A list with the following elements: `[10, 20, 25]`.
- A set that contains `10`, `20`, and `25`.
- Error: invalid syntax.

3) What's the result of `set([100, 200, 100, 200, 300])`?

- A list with the following elements: `[100, 200, 100, 200, 300]`.
- A set that contains `100`, `200`, and `300`.
- A set that contains `100`, `200`, `300`, another `100`, and another `200`.
- Error: invalid syntax.

Dictionary type

A ***dictionary*** is an unordered, indexed, mutable container. A dictionary is written with curly braces, indicating that elements are not ordered, rather than square brackets. Unlike list and tuple indexes, a dictionary index is any immutable type, such as an integer, string, or tuple. Ex:

- `dict = {'Messi': 10, 'Ronaldo': 7}` creates a dictionary with indexes 'Messi' and 'Ronaldo' and values 10 and 7.
- `{'Messi': 10, 'Ronaldo': 7}` and `{'Ronaldo': 7, 'Messi': 10}` are the same dictionary, since dictionaries are not ordered.
- `dict = {}` creates an empty dictionary.

A dictionary element is accessed by appending the index, within brackets, to the variable name. Since a dictionary is mutable, elements can be updated, added, and removed. Ex:

- `del dict['Rachel']` removes the element with key 'Rachel'.
- `dict['Amelia'] = 'A+'` either changes the value of an existing 'Amelia' element to 'A+' or adds a new 'Amelia' element.

The dictionary index is often called a ***key*** and the elements are often called ***key:value pairs***.

Dictionary. Full screen

- Click the double right arrow icon to restart the kernel and run all cells.
- Examine the code below.
- In cell [3], change `'banana'` to `'orange'`. Restart the kernel and run all cells to see the changes.
- Click "File", then "Download as". Download the Jupyter notebook as a PDF or HTML file.

```
In [1]: # Create the prices dictionary
        prices = {'apple': 1.99, 'orange': 1.49}

In [2]: # Add an element with key 'banana'
        prices['banana'] = 1.59
        prices

Out[2]: {'apple': 1.99, 'orange': 1.49, 'banana': 1.59}

In [3]: # Modify the banana element
        prices['banana'] = 1.69
        prices

Out[3]: {'apple': 1.99, 'orange': 1.49, 'banana': 1.69}

In [4]: # Remove the apple element
        del prices['apple']
        prices

Out[4]: {'orange': 1.49, 'banana': 1.69}
```


PARTICIPATION ACTIVITY 2.2.5: Modifying dictionaries.

1) Which statement adds 'pears' to the following dictionary?

```
prices = {'apples': 1.99,
'oranges': 1.49, 'kiwi': 0.79}
```

- `prices['pears'] = 1.79`
- `prices['pears': 1.79]`

2) Executing the following statements produces a KeyError:

```
prices = {'apples': 1.99,
'oranges': 1.49, 'kiwi': 0.79}
del prices['limes']
```

- True
- False

3) Executing the following statements adds a new entry to the dictionary:

```
prices = {'apples': 1.99,
'oranges': 1.49, 'kiwi': 0.79}
prices['oranges'] = 1.29
```

- True
- False

CHALLENGE ACTIVITY 2.2.1: Python data types.

537150.4174434.qx3zqy7

Start

- Create a list `exampleList1` with elements 0, 'b', and 16.
- Replace the first element with 'k'.
- Append 24 to `exampleList1`.
- Remove the third element from `exampleList1`.
- Combine `exampleList1` with the list `exampleList2`.

The code provided displays all results and creates the second list, `exampleList2`.

```
exampleList2 = ['fox', 101]

# Create a list with elements 0, 'b', 16
exampleList1 = # Your code goes here
print(exampleList1)

# Replace the first element with 'k'
# Your code goes here
print(exampleList1)

# Append 24 to the list
# Your code goes here
print(exampleList1)

# Remove the third element from the list
# Your code goes here
print(exampleList1)

```

1 2 3 4

Check Next level

2.3 Python functions

Learning goals

- Define function components.
- Identify scope of variable and function names.
- Identify keyword arguments, default parameters, and function parameters.
- Define and predict the results of pass-by assignment.

Function basics

Functions are either built into Python or defined by the programmer. A function definition consists of a header and a body.

The ***function header*** is a statement consisting of the `def` keyword, function name, parameters in parentheses, and a colon. A ***parameter*** is a variable that appears in the header and is used in the body. A function may have no parameters or multiple parameters separated by commas.

The ***function body*** follows the header and consists of an indented block of statements. The body may include return statements. A ***return*** statement terminates function execution. If the body has no return statements, execution terminates at the last statement in the body.

A ***function call*** consists of the function name followed by arguments, in parentheses, corresponding to parameters. An ***argument*** is any expression that evaluates to the type of the corresponding parameter. When the function is called, argument values are assigned to parameters and the function body is executed.

A function call returns the value of an expression specified in a return statement. A function with no return statement, or a return statement with no expression, returns the value None. Although a function returns exactly one value, the value may be a container type with multiple elements.

PARTICIPATION ACTIVITY 2.3.1: Function example.

```
def calcPizzaVolume(pizzaDiameter, pizzaHeight):
  piVal = 3.14159265
  pizzaRadius = pizzaDiameter / 2.0
  pizzaArea = piVal * pizzaRadius * pizzaRadius
  pizzaVolume = pizzaArea * pizzaHeight
  return pizzaVolume

print('12.0 x 0.3 inch pizza is', calcPizzaVolume(12.0, 0.3), 'cubic inches')

print('16.0 x 0.8 inch pizza is', calcPizzaVolume(16.0, 0.8), 'cubic inches')
```

```
12.0 x 0.3 inch pizza is 33.9 cubic inches.
16.0 x 0.8 inch pizza is 160.9 cubic inches.
```

Animation content:

Step1. The calcPizzaVolume function definition is highlighted. Step2. The first line of the calcPizzaVolume function definition (header) is highlighted. Step 3. The remaining lines of the calcPizzaVolume function definition (body) are highlighted. Step 4. The first calcPizzaVolume call is highlighted and the first printout appears in a console icon. Step 5. The second calcPizzaVolume call is highlighted and the second printout appears in the console icon.

Animation captions:

1. The calcPizzaVolume function calculates the volume of a pizza from diameter and height.
2. The function header includes parameters pizzaDiameter and pizzaHeight.
3. The function body computes and returns pizza volume.
4. The function call has arguments 12.0 and 0.3. Arguments are assigned to diameter and height parameters, and the function returns volume.
5. A second call calculates volume of a pizza with diameter 16.0 and height 0.8.

PARTICIPATION ACTIVITY 2.3.2: Function basics.

1) Which answer correctly defines two parameters `x` and `y` for the function definition `def calcVal ( _____ ):`?

   - `x; y`
   - `x y`
   - `x, y`

2) Which answer correctly passes two integer arguments for the function call `calcVal( _____ )`?

   - `99, 44 + 5`
   - `99 + 44`
   - `99 44`

3) Given a function definition `def calcVal(a, b, c):`, what value is `b` assigned with during the function call `calcVal(42, 55, 77)`?

   - Unknown
   - 42
   - 55

4) Given the function definition `def calcVal(a, b, c):` and variables `i`, `j`, and `k`, which are valid arguments in the call `calcVal( _____ )`?

- `i, j`
- `k, i + j, 99`
- `i + j + k`

Name scope

The ***scope*** of a variable or function name is the portion of a program in which the name is valid.

Scope begins with a definition. A variable definition is the variable's first assignment statement. A function definition is the function header. Scope ends as follows:

- If the definition is inside a function, the name is ***local***, and scope extends to the end of the function body.
- If the definition is outside all functions, the name is ***global***, and scope extends to the end of the file.

These scope rules have several consequences:

- Parameters are assigned with arguments inside a function. Therefore, parameters are local.
- Since function scope begins at the header, a function can call itself.
- A function call cannot precede the function header.
- If an inner function is defined in an outer function, the inner function is local.

If a variable is assigned both inside and outside a function, the assignments create two variables with the same name: one global and the other local. To assign a global variable inside a function, the variable must be declared global in the function body. In general, assigning global variables inside functions may cause unwanted results and should be avoided.

PARTICIPATION ACTIVITY 2.3.3: Global and local variables.

```
def changeName():
   employeeName = 'Juliet'

employeeName = 'Romeo'
changeName()
print('Employee name:', employeeName)
```

Employee name: Romeo

```
def changeName():
   global employeeName
   employeeName = 'Juliet'

employeeName = 'Romeo'
changeName()
print('Employee name:', employeeName)
```

Employee name: Juliet

Animation content:

Step 1. The first code fragment appears on the left. Step 2. The statement assigning Juliet to employeeName, inside the function, is highlighted. The statement assigning Romeo to

employeeName, outside the function, is highlighted. Step 3. The print statement is highlighted and Employee Name: Romeo appears in a console icon. Step 4. A copy of the code fragment appears on the right. A new statement global employeeName appears after the function header and the statement assigning employeeName with Juliet. Step 5. EmployeeName: Juliet appears in a new console icon below the code on the right.

Animation captions:

1. The changeName() function changes the employeeName variable.
2. employeeName is a local variable inside the function and a global variable in the main program.
3. The function changes the local variable, but the main program prints out the global variable.
4. employeeName is declared global inside the function.
5. Now the function changes the global variable.

PARTICIPATION ACTIVITY 2.3.4: Variable and function scope.

1) A local variable is defined inside a function, while a global variable is defined outside any function.
   - True
   - False

2) A local variable's scope extends from a function definition's ending colon ":" to the end of the function.
   - True
   - False

3) A global statement must be used to assign a global variable inside a function.
   - True
   - False

4) The function definition must appear before the function can be called.
   - True
   - False

Keyword arguments and default parameters

Arguments may be matched with parameters by either position or keyword:

- A ***positional argument*** is matched with a parameter by position in the function call and header. Ex: `pizzaVolume(12, 1.5)`.
- A ***keyword argument*** is preceded by a parameter name and matched with the parameter by name rather than position. Ex: `pizzaVolume(pizzaHeight=1.5, pizzaDiameter=12)`.

A ***default parameter*** is followed by a default value in the function header. If no argument is specified in a function call, the parameter is assigned with the default value. Ex: `def pizzaVolume(pizzaDiameter=12, pizzaHeight=1.5):`.

Keyword arguments and default parameters make code more readable and less error-prone. Keyword arguments are recommended when a function has many parameters. Default parameters are recommended when a parameter has a meaningful default value.

Keyword arguments and default parameters.

Full screen

- Click the double right arrow icon to restart the kernel and run all cells.
- Examine the code below.
- In cell [3], change the third argument to `False`. Restart the kernel and run all cells to see the changes.
- Click "File", then "Download as". Download the Jupyter notebook as a PDF or HTML file.

In [1]:
```
# Define function that prints full name
def printName(first, last, lastFirst=False):
    if lastFirst:
        print(last + ', ' + first)
    if not lastFirst:
        print(first + ' ' + last)
```

In [2]:
```
# Call with keyword arguments
printName(first='Dana', last='Patel', lastFirst=True)
```

```
Patel, Dana
```

In [3]:
```
# Call with positional arguments
printName('Dana', 'Patel', True)
```

```
Patel, Dana
```

In [4]:
```
# Call with both keyword and positional arguments
printName('Dana', 'Patel', lastFirst=True)
```

```
Patel, Dana
```

In [5]:
```
# Call with default value
printName('Dana', 'Patel')
```


PARTICIPATION ACTIVITY 2.3.5: Keyword arguments and default parameters.

Refer to the following function definition:

```
def splitCheck(amount=10,
               numPeople=2,
               taxPercentage=0.095,
               tipPercentage=0.18):
```

When entering answers, use the same number of significant digits as the default parameter values in the `splitCheck()` definition. Answer ERROR if an error occurs.

1) What value is passed as the `taxPercentage` argument in the function call `splitCheck(60.52, 5, 0.075, tipPercentage=0.18)`?

Check Show answer

2) What value is passed as the `numPeople` argument in the function call `splitCheck(taxPercentage=0.07, 60.52, 2, tipPercentage=0.18)`?

Check Show answer

3) What will the parameter `taxPercentage` be assigned for the following call? `splitCheck(amount=49.50, numPeople=3)`

Check Show answer

4) Write a statement that splits a $50 check among four people. Use the default tax percentage and tip amount.

Check Show answer

5) Write a statement that splits a $25 check among three people and leaves a 25% tip. Use the default tax rate.

Check Show answer

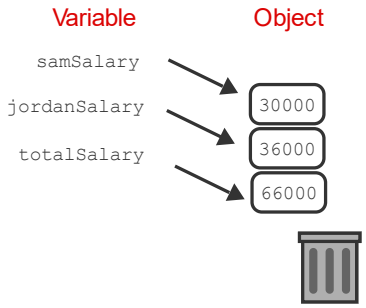
Function objects

In Python, every value is stored as an ***object***. An object has a value, a data type, and a unique identifier. The identifier is usually the memory address where the object's value is stored. An object is created whenever a statement containing a value is executed. Ex: `print('Hello')` creates an object with the value `'Hello'` and type string.

An assignment statement associates a variable with an object. A variable is always assigned with exactly one object, however an object can be assigned to many variables.

PARTICIPATION ACTIVITY 2.3.6: Assigning variables with objects.


```
samSalary = 25000
jordanSalary = 30000
samSalary = jordanSalary
jordanSalary = jordanSalary * 1.2
totalSalary = samSalary + jordanSalary
```



Animation content:

Step 1. The first line of code is highlighted. The variable name samSalary appears on the right with an arrow pointing to an object containing 25000. Step 2. The second line of code is highlighted. The variable name jordanSalary appears on the right with an arrow pointing to a new object containing 30000. Step 3. The third line of code is highlighted. The arrow from samSalary now points to the existing object containing 30000, and the object containing 25000 moves into a garbage can icon. Step 4. The fourth line of code is highlighted. The arrow from jordanSalary now points to a new object containing 36000. Step 5. The fifth line of code is highlighted. The variable name totalSalary appears on the right with an arrow pointing to a new object containing 66000.

Animation captions:

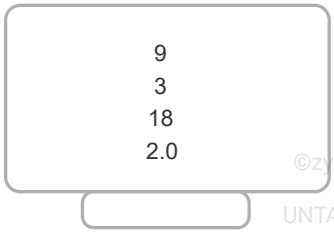
1. samSalary object is created.
2. jordanSalary object is created.
3. samSalary is assigned with jordanSalary. Memory allocated to the 25000 object is released.
4. jordanSalary is assigned with jordanSalary * 1.2.
5. totalSalary object is created and assigned with samSalary + jordanSalary.

Functions are also stored as objects. The object type is "function" and the object value is the function definition.

Function objects behave like variable objects. A function is assigned with a function object when the function is defined. A function can be reassigned with a new object. A function object can be passed to a function as an argument and returned from a function.

PARTICIPATION ACTIVITY 2.3.7: Function objects.

```
def plus(x, y):
   return x + y

def minus(x, y):
   return x - y

def times(x, y):
   return x * y

def divide(x, y):
   return x / y

def calculate(operation, x, y):
   return operation(x, y)

print( calculate(plus, 6, 3) )
print( calculate(minus, 6, 3) )
print( calculate(times, 6, 3) )
print( calculate(divide, 6, 3) )
```



Animation content:

Step 1. The first four function definitions are highlighted. Step 2. The fifth function definition, calculate(), is highlighted. Step 3. The first print statement is highlighted and 9 appears in a console icon. Step 4. The remaining print statements are highlighted and 3, 18, and 2.0 appear below the 9 in the console icon.

Animation captions:

1. The plus(), minus(), times(), and divide() functions implement arithmetic operations.
2. The operation parameter of calculator() is a function parameter.
3. The operation parameter is assigned with the plus function object.
4. The operation parameter is assigned with the minus, times, and divide function objects.

PARTICIPATION ACTIVITY 2.3.8: Function objects.

1) A function can return a function object.

- True
- False

2) A function can return itself. Ex: Function foo() can return the object assigned to the name foo.

- True
- False

3) The output of the following program is "meow":

```
def printCat():
    print('meow')

def printPig():
    print('oink')

printCat = printPig
printCat()
```

- True
- False

4) If func1() and func2() are defined functions, then the expression `func1 + func2` returns a valid value.

- True
- False

5) The expression `func1(func2())` passes the func2 function object as an argument to func1.

- True
- False

Pass by assignment

An object is either mutable or immutable. The value of a ***mutable*** object can be changed. The value of an ***immutable*** object cannot change. Objects of type list, set, and dictionary are mutable. Objects of type integer, float, Boolean, string, and tuple are immutable.

Variables are not inherently mutable or immutable. A variable can be assigned with a mutable or immutable object at any time. Ex: String objects are immutable, but `string = 'Hello'` may be followed by `string = 'Goodbye'`.

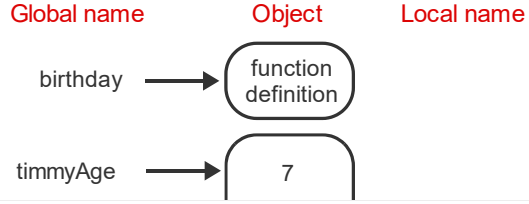
When a function is called, parameter names are assigned with argument objects. This mechanism is called ***pass by assignment***. Pass by assignment may affect global variables:

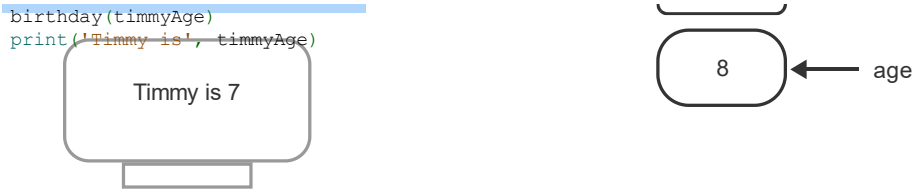
- If an argument object is mutable, the function can change the object, so the argument can change outside the function.
- If the argument object is immutable, the function cannot change the object. The function can reassign the parameter, but the argument cannot change.

If a parameter has a mutable default object, the function may change the default object. This change persists across function calls, which is usually unintended and undesirable. Usually, parameter defaults should be immutable.

PARTICIPATION ACTIVITY 2.3.9: Pass by assignment.


```
def birthday(age):
   '''Celebrate birthday!'''
   age = age + 1

timmyAge = 7
```





Animation content:

Step 1. The first line of code is highlighted. The name birthday appears under the caption Global variables. An arrow points from birthday to an object containing function definition. Step 2. The statement timmyAge = 7 is highlighted. The name timmyAge appears under the caption Global variables. An arrow points from timmyAge to an object containing 7. The statement birthday(timmyAge) is highlighted, then the birthday() function header is highlighted. The name age appears under the caption Local variables. An arrow points from age to the object containing 7. Step 3. The statement age = age + 1 in the function body is highlighted. The arrow from age now points to a new object containing 8. Step 4. The print statement is highlighted and Timmy is 7 appears in a console icon.

Animation captions:

1. birthday() is a global function. The function definition is stored as an object.
2. The function call assigns parameter age with the timmyAge object.
3. Assigning parameter age with a new object does not change argument timmyAge.
4. Since timmyAge has not changed, "Timmy is 7" is displayed.

Parameter passing alternatives

Many programming languages pass parameters by value or by reference rather than by assignment.

Pass by value *assigns parameters with a copy of argument objects. The function cannot access the original objects and therefore cannot change the arguments.*

Pass by reference *assigns parameters with the original argument objects, like pass by assignment. However, the two are not identical. In pass by assignment, reassigning a parameter assigns a new object. The argument object does not change. In pass by reference, reassigning a parameter changes the argument object.*

Pass by assignment is sometimes called ***pass by object reference****, which is not the same as pass by reference.*

PARTICIPATION ACTIVITY 2.3.10: Pass by assignment.

What does the following code print?

1)
```
def modify(numList):
    numList[1] = 99

list = [10, 20, 30]
modify(list)
print(list)
```

- [10, 20, 30]
- [10, 99, 30]
- An error message

2)
```
def modify(string):
    string[0] = 'C'

string = 'Hello'
modify(string)
print(string)
```

- Hello
- Cello
- An error message

3)
```
def modify(string):
    string = 'Cello'

string = 'Hello'
modify(string)
print(string)
```

- Hello
- Cello
- An error message

CHALLENGE ACTIVITY 2.3.1: Python functions.

537150.4174434.qx3zqy7

Start

Print the return value of the function `findDifference()` for x=4 and y=2.

The code provided defines the function.

```
1 # Defines the function findDifference
2 def findDifference(x, y):
3     result = x - y
4     return result
5
6 # Prints the output of the function findDifference() for x=4 and y=2
7 print(# Your code goes here)
```

1 2 3 4

Check Next level

2.4 Data science packages

Learning goals

- Use Jupyter to import a package in Python.
- Explain how to install a Python package.
- Identify and compare data science packages for Python.

Python packages

A Python ***package*** is a collection of related functions, classes, and objects that are loaded from the hard drive to extend the functionality of Python.

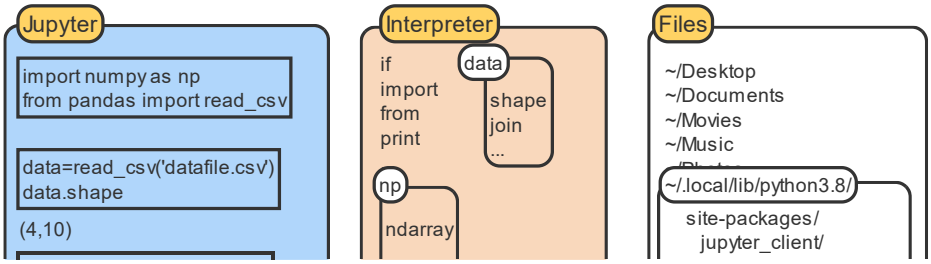
A package is loaded using the `import` command. Ex: `import pandas` loads the pandas package.

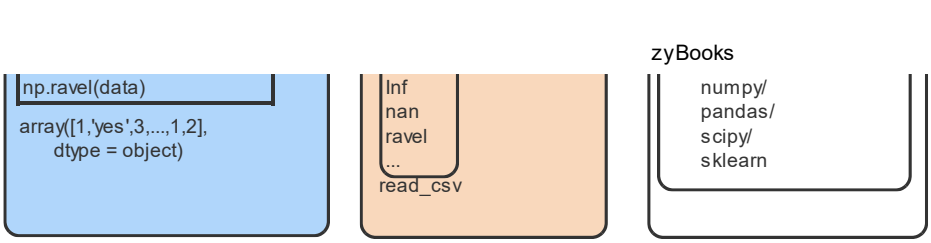
A package can be aliased at import by using `as`. Ex: `import pandas as pd`. Aliasing allows for shorter names for frequently used packages.

Functions and objects in an imported package are accessed via dot notation. Ex: `pd.read_csv('datafile.csv')` reads the file 'datafile.csv' in as a `pandas` DataFrame.

A single function can be imported from a package using the `from` command. Ex: `from sklearn.preprocessing import StandardScaler`.

PARTICIPATION ACTIVITY 2.4.1: Loading a package.





Animation content:

Step 1: When a Python interpreter is started, the only things in memory are default keywords and functions for the programming language.

Two sections of the screen appear. One is labeled 'Python environment' and is blank with the exception of a blank rectangle. The second is labeled 'Memory' and has the words {if, import, from, print, ...} listed.

Step 2: An import statement loads the package from the hard drive and is stored by the interpreter under the alias.

The commands "import numpy as np
from pandas import read_csv" appear in the empty rectangle in the 'Python environment'.

A third region appears that is labeled 'Files'. Listed in this region are various folder paths.
A green box highlights "import numpy as np".
The last folder expands to '~/.local/lib/python3.8/. In that folder there are the folders jupyter_client, numpy, pandas, scipy, and sklearn.
A box labeled 'np' that contains {ndarray, ndarray.shape, ndarray.ravel, inf, nan, ...} flies from the numpy folder to the memory region.

Step 3: A from statement follows the same process, but the object requested is held in the memory space

The green box moves down to the line "from pandas import read_csv". "read_csv" flies into the 'Memory' region from the pandas folder
The green box disappears and a new empty rectangle appears in the Python environment.

Step 4:A called function or variable is accessed from memory by the interpreter, whether user-defined or package-defined.

The code "data=read_csv('datafile.csv')
data.shape" appears inside the empty rectangle.

A green rectangle highlights "data=read_csv('datafile.csv')".
"read_csv" flies in from "Memory" and "datafile.csv" flies in from the Files region to the locations in the code. "data" flies to the "Memory" region where "data" has a box that contains {shape, join, ...}

The green rectangle moves down to the line "data.shape" "data" and "shape" fly from 'Memory' to the location in code and the output "(4,10)" appears below the rectangle containing the code. The green rectangle disappears and a new empty rectangle appears below the output.

Step 5:Functions within packages are accessed using dot notation starting with the alias for the package.

The code "np.ravel(data)" appears in the empty rectangle. "data" and "ravel" fly in from their locations in 'Memory" and the output "array([1,'yes',3,...,1,2], dtype = object)" appears.

Animation captions:

1. When a Python interpreter is started, the only things stored are default keywords and functions for the programming language.
2. An import statement loads the package from the hard drive and is stored by the interpreter under the alias.
3. A from statement follows the same process, but the object requested is stored outside of any other object.
4. A called function or variable is accessed from memory by the interpreter, whether user-defined or package-defined.
5. Functions within packages are accessed using dot notation starting with the alias for the package.

PARTICIPATION ACTIVITY 2.4.2: Importing packages in Python.

Provide the command that does the described action.

1) Import the package `numpy`

Check Show answer

2) Import the function `read_csv` from the package `pandas`

Check Show answer

3) Import the package `matplotlib.pyplot` under the alias `plt`

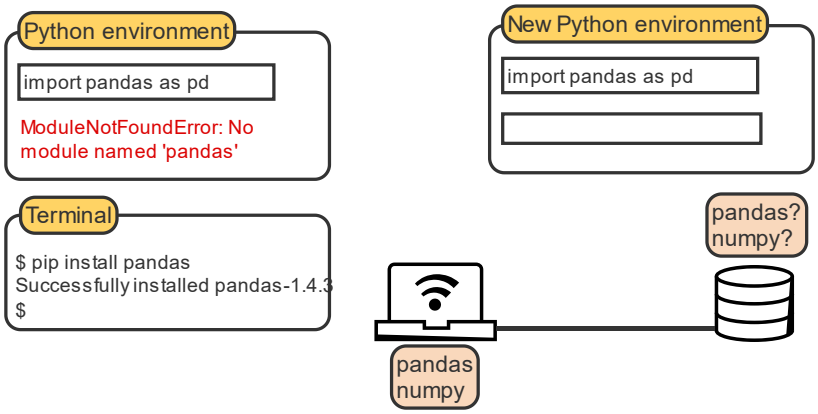
Check Show answer

Obtaining packages

Too many packages for Python exist to be easily installed with the Python interpreter. Packages are instead downloaded when requested from servers called repositories. A ***dependency*** is when a package requires the installation of another package to work. A ***package manager*** installs, updates, and removes packages while ensuring dependencies are maintained.

The two most common package managers for Python are `pip` and `conda`. `pip` comes installed with Python and can be accessed via a terminal. A ***terminal*** is a text-based interface on the computer where actions are communicated via typed commands. Terminals can be started in a local installation of Jupyter or with the operating system outside Jupyter.

PARTICIPATION ACTIVITY 2.4.3: Installing a package.



Animation content:

Step 1: An error is returned when a package is not found in Python's path.

A region labeled 'Python environment' is on the screen with an empty rectangle inside.
The code "import pandas as pd" appears in the empty rectangle.
The text "ModuleNotFoundError: No module named 'pandas'" appears below the rectangle in red.

Step 2: The user then installs the package via a package manager in a terminal. The package manager checks if other packages need to be installed to satisfy any dependencies.

A region labeled "Terminal" appears. The command "pip install pandas" appears.

Step 3: The user's computer requests the package and any dependencies from the repository and the repository sends the files needed back to the computer.

A laptop and a server appear connected with a line. The words "pandas? numpy?" go from the laptop to the server. The words "pandas numpy" go from the server to the laptop.

Step 4: Now the package is available in a new (or restarted) Python environment.

A region labeled 'New Python environment' is on the screen with an empty rectangle inside.
The code "import pandas as pd" appears in the empty rectangle.
A new empty rectangle appears.

Animation captions:

1. An error is returned when a package is not found in Python's path.
2. The user then installs the package via a package manager in a terminal. The package manager checks if other packages need to be installed to satisfy any dependencies.
3. The user's computer requests the package and any dependencies from the repository, and the repository sends the files needed back to the computer.
4. Now the package is available in a new (or restarted) Python environment.

PARTICIPATION ACTIVITY 2.4.4: Packages in Python.

Match the term to the definition.

If unable to drag and drop, refresh the page.

Package manager **Dependency** **Package**

	A collection of classes and functions that can be imported into a Python program.
	A package that must be installed for another package to work
	A program that installs, removes, and updates packages while maintaining any dependencies between packages

Reset

Common data science packages

Many Python packages exist for data manipulation, visualization, and modeling. Different packages provide different functionality, so a data science project will use multiple packages.

Table 2.4.1: Common data science packages in Python.

Import name	Common alias	Description
`numpy`	`np`	NumPy includes functions and classes that aid in numerical computation. `NumPy` is used in many other data science packages.
`pandas`	`pd`	pandas provides methods and classes for tabular and time-series data.
`sklearn`	`sk`	scikit-learn provides implementations of many machine learning algorithms with a uniform syntax for preprocessing data, specifying models, fitting models with cross-validation, and assessing models.
`matplotlib.pyplot`	`plt`	matplotlib allows the creation of data visualizations in Python. The functions mostly expect `NumPy` arrays.
`seaborn`	`sns`	seaborn also allows the creation of data visualizations but works better with pandas DataFrame.
`scipy.stats`	`sp.stats`	SciPy provides algorithms and functions for computing problems that arise in science,

		engineering and statistics. `scipy.stats` provides the functions for statistics.
`statsmodels`	`sm`	statsmodels adds functionality to Python to estimate many different kinds of statistical models, make inferences from those models, and explore data.

Using data science packages in Jupyter notebooks.

Full screen

The Python code below uses the packages above to explore and model with data from three species of hawk collected in Iowa.

- Click the double right arrow icon to restart the kernel and run all cells.
- Examine the code below. Pay attention to the comments.
- Change the cutoff until the worst of the outliers have smaller values.
- Increase the `max_depth` in the random forest classifier to improve the number of correct classifications.

In [1]:
```
# Import packages

## Most coding styles require package imports at the top of the notel
## This style prevents running much of a notebook to find a package r
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.model_selection import train_test_split
```

Load dataset

In [2]:
```
# Load the hawks dataset
hawks = pd.read_csv('hawks.csv')
```

This dataset includes data about hawks at a nature preserve in Iowa.

- Species - CH:Cooper's Hawk, SS:Sharp-shinned Hawk, RT:Redtail Hawk
- Age - A:Adult, I:Immature
- Wing - length of the primary wing feather in mm
- Weight - Body weight in g
- Culmen - Length along the top of the bill from tip to face in mm
- Hallux - Length in mm of the killing talon
- Tail - Aggregate measurement related to the length of the tail in mm

PARTICIPATION ACTIVITY 2.4.5: Choosing data science packages in Python.

Select the best package to do the task described.

1) Read, investigate, and manipulate datasets
   - matplotlib
   - pandas
   - scikit-learn

2) Visualize a dataset

- NumPy
- pandas
- seaborn

3) Create and evaluate a machine learning model from a prepared dataset

- NumPy
- SciPy
- scikit-learn

2.5 NumPy package

Learning goals

- Define NumPy arrays.
- Define NumPy array functions.
- Use NumPy functions to find the dimensions and shape of an array.
- Use NumPy functions to create and modify an array.

Arrays

The ***NumPy*** package provides mathematical functions such as polynomial computations, matrix algebra, and statistical analysis. NumPy is pronounced "num-pie". Data scientists often apply these functions to one- or two-dimensional data, represented with the NumPy ndarray type.

The ***ndarray*** type is an ordered, indexed, and mutable container. Elements are of any type; however, all elements must be of the same type. An ndarray object is called an ***array*** and is created by the array() function.

An array may have zero, one, or many dimensions:

- A zero-dimensional array consists of a scalar object. Ex: 2.
- A one-dimensional array consists of a container of scalars. Ex: [2, 4, 6, 8].
- A two-dimensional array consists of a container of containers of scalars. Ex: [[2, 4, 6, 8], [12, 14, 16, 18]].

An N-dimensional array has N levels of nested containers. At each level, all containers should have the same ***size***, or number of elements. The ***shape*** of an array is a tuple of level sizes. Ex: The shape of [[2, 4, 6, 8], [12, 14, 16, 18]] is (2, 4). The shape tuple is stored in the array.shape attribute.

Array elements are accessed with index notation. Ex: `array[2, 5]` returns the element in the third row and sixth column of a two-dimensional array.

Array literals

A literal such as `[2, 4, 6, 8]` is actually a list, not an array. Formally, an array literal is written as `array([2, 4, 6, 8])`. In the interest of brevity, this section denotes array literals as `[2, 4, 6, 8]` when the type is clear from context.

Arrays.

Full screen

- Click the double right arrow icon to restart the kernel and run all cells.
- Examine the code below.
- In cell [2], add `[100, 200, 300]` to the list inside the `np.array()` function. Restart the kernel and run all cells to see the changes.
- Click "File", then "Download as". Download the Jupyter notebook as a PDF or HTML file.

©zyBooks 03/17/24 16:51 2087217
Biniam abebe
UNTADTA5340OrhanSpring8wk22024

```
In [1]:  # Load NumPy package
         import numpy as np

In [2]:  # Create two-dimensional array
         myArray = np.array([[10, 20, 30], [210, 220, 230]])

In [3]:  # Display the array
         myArray

Out[3]: array([[ 10,  20,  30],
               [210, 220, 230]])

In [4]:  # Display the shape
         myArray.shape

Out[4]: (2, 3)

In [5]:  # Display the size
         myArray.size

Out[5]: 6

In [6]:  # Access the second row and third column
         myArray[1, 2]
```


PARTICIPATION ACTIVITY 2.5.1: Array shape.

Match the shape to the array.

If unable to drag and drop, refresh the page.

(4, 2) **( )** **(1)** **(2, 4)** **(2, 2, 3)** **(4)** **invalid array**

	`['a', 'b', 'c', 'd']`
	`[ [ [1, 2, 9], [3, 2, 6] ], [ [8, 8, 4], [9, 8, 7] ] ]`
	`[ [1, 2], [3, 4], [5, 6], [7, 8] ]`
	`3.1415`
	`[ [10, 20, 30, 40], [50, 60, 70] ]`
	`[ [10, 20, 30, 40], [50, 60, 70, 80] ]`

```
[3.1415]
```

Reset

Array functions

In this section, tables of functions follow several conventions:

- The tables include all required parameters. The tables also include important optional parameters, but exclude infrequently used optional parameters.
- `NumPy` functions are written with the prefix 'numpy' or an alias. The tables omit this prefix. Ex: `sort(array)` stands for `numpy.sort(array)`.
- Many `NumPy` functions have equivalent `ndarray` methods. Ex: `sort(array)` is equivalent to `array.sort()`. The tables document functions only and omit equivalent methods.

The table below describes functions that create, update, and sort arrays. Additional `NumPy` functions are described elsewhere in this material and in the NumPy API reference.

Some functions have an ***axis*** parameter. An axis is a dimension. In a two-dimensional array, axis 0 refers to rows and axis 1 refers to columns. Ex: `sort(array, axis=1)` sorts column values (`axis=1`) within each row.

Table 2.5.1: Array functions.

Function	Parameters	Description

`array()`	`object`
	`dtype=None`
	`ndmin=0`
		Returns an array constructed from `object`. `object` must be a scalar or an ordered container, such as tuple or list. The array element type is inferred from `object` unless a `dtype` is specified. `ndim` is the minimum number of array dimensions.

`delete()`	`arr`
	`object`
	`axis=None`
		Deletes a slice of input array `arr`. `axis` is the axis along which to remove a slice. `obj` is the index of the slice along the axis.

`full()`	`shape`
	`fill_value`
	`dtype=None`
		Returns an array filled with `fill_value`. The `shape` tuple specifies array shape. `dtype` specifies the array type. If `dtype=None`, the type is inferred from `fill_value`.

`insert()`	`arr`
	`object`
	`values`
	`axis=None`
		Inserts array `values` to input array `arr`. `axis` is the axis along which to insert. `obj` is the index before which `values` is inserted.

`zeros()`	`shape` `dtype=float`	Returns an array filled with zeros. The `shape` tuple specifies array shape. `dtype` specifies the array type.
`ones()`	`shape` `dtype=None`	Returns an array filled with ones. The `shape` tuple specifies array shape. `dtype` specifies the array type. If `dtype=None`, the type is float64.

`sort()`	`a` `axis=-1`	Sorts array `a` along `axis`. The default `axis=-1` sorts along the last axis in `a`. `axis=None` flattens `a` before sorting.

Array functions.

Full screen

- Click the double right arrow icon to restart the kernel and run all cells.
- Examine the code below.
- In cell [3], change the value of the `axis` parameter to `1` and the index value to 0. Restart the kernel and run all cells to see the changes.
- Click "File", then "Download as". Download the Jupyter notebook as a PDF or HTML file.

```
In [1]:
# Load NumPy package
import numpy as np
```

```
In [2]:
# Create array
array1 = np.array([['a', 'a'], ['b', 'b'], ['c', 'c']])
```

```
In [3]:
# Insert 'z's in row 3
np.insert(array1, 3, 'z', axis=0)
```

```
Out[3]: array([['a', 'a'],
               ['b', 'b'],
               ['c', 'c'],
               ['z', 'z']], dtype='<U1')
```

```
In [4]:
# Insert 0s in column 1
np.insert(array1, 1, '0', axis=1)
```

```
Out[4]: array([['a', '0', 'a'],
               ['b', '0', 'b'],
               ['c', '0', 'c']], dtype='<U1')
```

```
In [5]:
# Delete row 1
np.delete(array1, 1, axis=0)
```



```
Out[5]: array([['a', 'a'],
```

PARTICIPATION ACTIVITY 2.5.2: Array functions.

Assume `NumPy` is loaded under the alias `np`. This code initializes and prints `array`:

```
array = np.array( [ ['a', 'a'], ['b', 'b'], ['c', 'c'] ] )
print(array)
```

```
[['a' 'a']
 ['b' 'b']
 ['c' 'c']]
```

The following questions modify `array` and print the result. Fill in the blank to complete the code.

1)
```
array2 = np._____(array, 0,
'z', axis=1)
print(array2)
```

```
[['z' 'a' 'a']
 ['z' 'b' 'b']
 ['z' 'c' 'c']]
```

Check Show answer

2)
```
array2 = np.insert(array, 0,
'z', _____)
print(array2)
```

```
[['z' 'z']
 ['a' 'a']
 ['b' 'b']
 ['c' 'c']]
```

Check Show answer

3)
```
array2 = np._____(array, 1,
axis=0)
print(array2)
```

```
[['a' 'a']
 ['c' 'c']]
```

Check Show answer

4)
```
array2 = np.sort(array, ____)
print(array2)
```

```
['a' 'a' 'b' 'b' 'c' 'c']
```

Check Show answer

©zyBooks 03/17/24 16:51 2087217
Biniam abebe
UNTADTA5340OrhanSpring8wk22024

Shape functions

Occasionally, data scientists must change the shape of an array. Ex: Two-dimensional data might be structured as one-dimensional in a CSV file. Reading the file creates a one-dimensional array, which must be restructured to two dimensions.

Reshaping an array changes array dimensions while retaining array data. Ex: `[[1, 3, 5], [2, 4, 6]]` might be reshaped to `[[1, 2], [3, 4], [5, 6]]`. ***Flattening*** an array reshapes an N-dimensional array to one dimension.

Reshaping a two-dimensional array prioritizes element order in either rows or columns. ***Row-major*** order prioritizes row order. ***Column-major*** order prioritizes column order. Ex: Flattening `[[1, 3, 5], [2, 4, 6]]` returns:

- `[1, 3, 5, 2, 4, 6]` in row-major order.
- `[1, 2, 3, 4, 5, 6]` in column-major order.

Row- and column-major order applies to arrays of three or more dimensions. Row-major reshaping prioritizes element order in the first dimension, then the second, and so on. Column-major reshaping prioritizes dimensions in reverse order.

Some shape functions have an `order` parameter. `order='C'` specifies row-major order. `order='F'` specifies column-major order. 'C' and 'F' stand for the languages C and FORTRAN, which store arrays in row- and column-major order, respectively.

Table 2.5.2: Shape functions.

Function	Parameters	Description
`ravel()`	`a` `order='C'`	Returns flattened array `a`.

`reshape()`	`a`
	`newshape`
	`order='C'`
		Returns an array with the same data as `a` but a different shape. `newshape` is an integer or tuple of integers that specifies the new shape. The new shape must have the same number of elements as the original shape.

`resize()`	`a` `newshape`	Returns an array with the same data as `a` but a different shape. `newshape` is an integer or tuple of integers that specifies the new shape. The new and original arrays may have a different number of elements. If the new array is larger than the original array, then the new array is filled with repeated copies of `a`.
`transpose()`	`a`	Returns a transposed copy of array `a`. Zero- and one-dimensional arrays are not changed. Equivalent to the attribute `array.T`.

Shape functions.

Full screen

- Click the double right arrow icon to restart the kernel and run all cells.
- Examine the code below.
- In cell [6], change the shape of the array to (6,1). Restart the kernel and run all cells to see the changes.
- Click "File", then "Download as". Download the Jupyter notebook as a PDF or HTML file.

```
In [1]: # Load the NumPy package
        import numpy as np

In [2]: # Create an array
        array1 = np.array([[1, 2], [3, 4]])
        array1

Out[2]: array([[1, 2],
               [3, 4]])

In [3]: # Return a flattened array
        np.reshape(array1, (1, 4))

Out[3]: array([[1, 2, 3, 4]])

In [4]: # Create another array
        array2 = np.array([[1, 2], [3, 4], [5, 6]])
        array2

Out[4]: array([[1, 2],
               [3, 4],
               [5, 6]])

In [5]: # Return a flattened array
        np.ravel(array2)
```


PARTICIPATION ACTIVITY 2.5.3: Shape functions.

Variable `array` is assigned with `[ [1, 2, 3, 4], [5, 6, 7, 8] ]`. Match the function call with the resulting array.

If unable to drag and drop, refresh the page.

`ravel(array, order='C')` `ravel(array, order='F')`

`transpose(array)` `resize(array, (2, 5))`

`reshape(array, (2, 2, 2))`

`[ [ [1, 2], [3, 4] ], [ [5, 6], [7, 8] ] ]`

`[1, 5, 2, 6, 3, 7, 4, 8]`

`[ [1, 5], [2, 6], [3, 7], [4, 8] ]`

`[1, 2, 3, 4, 5, 6, 7, 8]`

```
[ [1, 2, 3, 4, 5], [6, 7, 8, 1, 2] ]
```

Reset

Math operators and functions

`NumPy` offers a rich library of mathematical operators and functions. Many work with arrays:

- An ***arithmetic operator***, such as `+ - * / %`, may have array operands. The operator is applied to element pairs and returns an array.
- A ***simple function***, such as `sqrt()` and `log()`, may have an array argument. The function is applied to each element and returns an array.

Some functions are designed for arrays rather than scalars:

- An ***aggregate function***, such as `min()`, `max()`, `median()`, and `var()`, returns one value for an entire array or array slice. Many aggregate functions are statistical. Ex: `std(array)` returns the standard deviation of `array` elements. `var(array)` returns the variance of `array` elements.
- A ***matrix function*** interprets arrays as mathematical matrices and implements matrix algebra. Ex: `dot(array1, array2)` returns the dot product of `array1` and `array2`. `cross(array1, array2)` returns the cross product.

Aggregate and matrix functions also work with scalar arguments, since a scalar is a zero-dimensional array.

Table 2.5.3: Math operator and function examples.

	Expression	Description
Arithmetic operators	`array1 + array2`	Element-wise addition
	`array1 - array2`	Element-wise subtraction
	`array1 * array2`	Element-wise multiplication
	`array1 / array2`	Element-wise division
Simple functions	`sqrt(array1)`	Square root of array elements
	`log(array1)`	Logarithm of array elements
	`sin(array1)`	Sine of array elements
Aggregate functions	`max(array1)`	Maximum of array elements
	`median(array1)`	Median of array elements
	`std(array1)`	Standard deviation of array elements
	`var(array1)`	Variance of array elements
Matrix functions	`dot(array1, array2)`	Dot product array1 rows with array2 columns

	`matmul(array1, array2)`	Matrix product of array1 and array2
	`cross(array1, array2)`	Cross product of array1 and array2

The + operator

On list operands, + implements concatenation, not addition. Ex: `[5, 5, 5] + [1, 2, 3]` *returns* `[5, 5, 5, 1, 2, 3]` *rather than* `[6, 7, 8]`.

Math operators and functions.

Full screen

- Click the double right arrow icon to restart the kernel and run all cells.
- Examine the code below.
- In cell [8], change `np.sqrt` to `np.log`. Restart the kernel and run all cells to see the changes.
- Click "File", then "Download as". Download the Jupyter notebook as a PDF or HTML file.

```
In [1]: # Load NumPy package
        import numpy as np

In [2]: # Create array1
        array1 = np.array([[1, 2], [3, 4]])
        array1

Out[2]: array([[1, 2],
               [3, 4]])

In [3]: # Create array2
        array2 = np.array([[5, 6], [7, 8]])
        array2

Out[3]: array([[5, 6],
               [7, 8]])

In [4]: # Add array1 and array2
        array1 + array2

Out[4]: array([[ 6,  8],
               [10, 12]])

In [5]: # Subtract array1 and array2
        array1 - array2
```


PARTICIPATION ACTIVITY 2.5.4: Math operators and functions.

1) All array functions may take a scalar argument.
   - True
   - False

2) Comparison operators such as <, >, and == work with array operands.
   - True
   - False

3) An aggregate function always returns exactly one value.
   - True
   - False

4) The variance function `var(array, axis)` is a simple function.

- True
- False

CHALLENGE ACTIVITY 2.5.1: Using NumPy.

537150.4174434.qx3zqy7

Start

Create a 2-dimensional array containing the arrays [4, 9], [17, 18], [15, 5], [16, 19], [13, 2].

```
# Load the necessary package
import numpy as np

# Create an array
myArray = # Your code goes here

# Print the array
print(myArray)
```

1 2 3 4 5

Check Next level

2.6 pandas package

Learning goals

- Define the `DataFrame` object type.
- Compare `pandas` dataframes and `NumPy` arrays.
- Use the `DataFrame` constructor to create dataframes.
- Use labels, indices, and slices to subset dataframes.
- Use comparison and logical operators to subset dataframes.

- Define `pandas` methods for modifying dataframes.

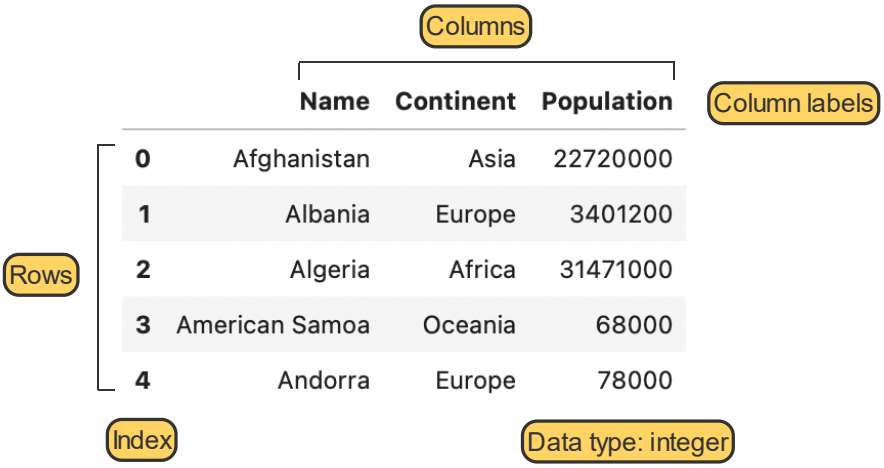
Dataframes

pandas is a Python package that stores and manipulates datasets. `pandas` represents datasets with the ***DataFrame*** type. In this material, ***dataframe***, in lowercase, refers to a DataFrame object.

A dataframe consists of rows and columns, which represent dataset instances and features, respectively. Rows and columns are identified by integer or string ***labels***. The set of row labels is called the ***index*** and the set of column labels is called ***columns***. Usually, row labels are automatically generated integers, and column labels are manually specified strings.

All values in a column must have the same type, but different columns may have different types. Commonly used data types for columns are integer, float, or string.

PARTICIPATION ACTIVITY 2.6.1: Dataframes.



Animation content:

A dataframe with five rows and three columns. Row labels are 0 through 4. Column labels are Name, Continent, and Population. The index gives the row labels, 0 through 4. The data type of population is integer.

Animation captions:

1. The dataframe contains information about different countries.
2. Columns represent features. Column labels are usually strings.
3. Rows represent instances. Row labels, called the index, are usually automatically generated integers.
4. Each column has a data type. The type of the Population column is integer.

PARTICIPATION ACTIVITY 2.6.2: Dataframes.

Match the term to the description.

If unable to drag and drop, refresh the page.

columns dataframe label index DataFrame

	column labels
	DataFrame object
	row or column identifier
	data type
	row labels

Reset

Dataframe vs. array

The `NumPy` package supports array objects.

A dataframe is similar to an array because:

- Both are indexed, ordered, and mutable containers.
- Both represent data in multiple dimensions, called ***axes***.
- Both have a ***shape***, a tuple of integers representing the number of elements along each axis.

However, dataframes and arrays differ in several ways:

- Dataframes are always two-dimensional. Arrays may have zero, one, or many dimensions.
- Different dataframe columns may have different types. All array values have the same type.
- Dataframe labels may be integers, strings, or other types. Array indexes are integers only.

Dataset features usually have string names and often have different types. For these reasons, datasets are usually implemented as dataframes in `pandas` rather than `NumPy` arrays.

Constructing dataframes. Full screen

- Click the double right arrow icon to restart the kernel and run all cells.
- Examine the code and text below.
- Change the column labels in cell [2]. Restart the kernel and run all cells to see the changes.
- Click "File", then "Download as". Download the Jupyter notebook as a PDF or HTML file.

In [1]:
```
# Load the pandas and numpy packages
import pandas as pd
import numpy as np
```

In [2]:
```
# Create dataframe with pandas DataFrame() constructor
dataframe = pd.DataFrame(
    data=[['abc', 3.3, 28, True],
          ['xyz', -0.55, 0, False]],
    columns=['Label1', 'Label2', 'Label3', 'Label4'],
    index=[0, 1],
)
```


In [3]:
```
# Display the dataframe
dataframe
```

Out[3]:

	Label1	Label2	Label3	Label4
0	abc	3.30	28	True
1	xyz	-0.55	0	False

In [4]:
```
# Display the dataframe's shape
dataframe.shape
```

Out[4]: (2, 4)

PARTICIPATION ACTIVITY 2.6.3: Constructing a dataframe.

Complete the following statement:

```
country = pandas.__A__(
    __B__=[ ['China', 'Asia', 9572900],
            ['Bangladesh', 'Asia', 143998],
            ['Brazil', 'South America', 8547403],
            ['Norway', 'Europe', 358207] ],
    __C__=['Country', 'Continent', 'Population'],
    __D__=[0, 1, 2, 3] )
print(country)
```

```
      Country      Continent. Population
0       China           Asia     9572900
1  Bangladesh           Asia      143998
2      Brazil  South America     8547403
3      Norway         Europe      358207
```


1) What is A?

Check Show answer

2) What is B?

Check Show answer

3) What is C?

Check Show answer

4) What is D?

Check Show answer

Subsetting data

Subsetting data involves choosing specific rows and columns from a dataframe according to labels, indices, and slices.

A single column can be selected by using the label of the desired column. Ex: Using the country dataset assigned to the variable `country`, the Population column can be selected using the `country['Population']` or `country.Population`. Multiple columns can also be selected by using an array of strings. Ex: `country[['Name', 'Population']]`

The `iloc(x,y)` method for a dataframe is used to select an individual element using an index location, where `x` is the row and `y` is the column. Ex: `country.iloc[0,1]` returns the element in row 0 and column 1. The colon character `:` is used in slice notation to select multiple rows or columns. Ex: `country.iloc[:5,1:3]` returns rows before row 5 and columns 1 thru 2.

The `loc(x,y)` method can also be used to subset data, but `y`, in this case, is an array of column labels, instead of an integer or a range of integers. Ex: Both `country.iloc[:7,1:3]` and `country.loc[:6,['Continent','Population']]` give the same results.

Table 2.6.1: Slice notation.

Notation	Description
`a:b`	index values from `a` to `b-1`
`:b`	index values before `b`
`a:`	index values from `a` onwards

For the `loc()` method, the index `b` is included in the resulting output.

Series and dataframes

`pandas` supports both series and dataframe objects. A series is a one-dimensional data structure that contains a list with an associate index. In contrast, a dataframe has a two-dimensional data structure that can contain more than one series. Specific methods in `pandas` return either a series or a dataframe.

Ex: `country[['Population']]` returns a dataframe, but `country['Population']` and `country.Population` return a series.

Subsetting data.

Full screen

- Click the double right arrow icon to restart the kernel and run all cells.
- Examine the code and text below.
- Change the values of the indices in cell [9] to display the first 10 rows and first two columns. Restart the kernel and run all cells to see the changes.
- Click "File", then "Download as". Download the Jupyter notebook as a PDF or HTML file.

©zyBooks 03/17/24 16:51 2087217
Biniam abebe
UNTADTA5340OrhanSpring8wk22024

```
In [1]: # Load the pandas package
        import pandas as pd
```

```
In [2]: # Load the country.csv data
        country = pd.read_csv('country_subset.csv')
```



```
In [3]: # Display the country dataframe
        country
```

Out[3]:

	Name	Continent	Population
0	Afghanistan	Asia	22720000
1	Albania	Europe	3401200
2	Algeria	Africa	31471000
3	American Samoa	Oceania	68000
4	Andorra	Europe	78000
...	...	...	...
234	Western Sahara	Africa	293000
235	Yemen	Asia	18112000
236	Yugoslavia	Europe	10640000
237	Zambia	Africa	9169000
238	Zimbabwe	Africa	11669000

PARTICIPATION ACTIVITY 2.6.4: Subsetting data.

1) Which statement selects the column `Name` from the `country` dataframe and returns another dataframe?

- `country.Name`
- `country['Name']`
- `country[['Name']]`

2) What `pandas` object is returned by the statement `country[['Name', 'Continent']]`?

- `Series`
- `DataFrame`

3) What statement returns the first 10 rows of the `country` dataframe?

- `country[1:10]`
- `country[0:10]`
- `country[[0:10]]`

4) What statement returns the first 5 rows of the dataframe and the columns `Name` and `Population`?

- `country.loc[0:4, ['Name', 'Population']]`
- `country.iloc[0:4, ['Name', 'Population']]`
- `country[0:4,['Name', 'Population']]`

Subsetting data using comparison and logical operators

Comparison and logical operators can be used to subset data. When these operators are used, only rows for which the expression is true will be returned. Ex: `country[country['Population'] > 100000]` will display rows whose `'Population'` column values are greater than 100,000.

Table 2.6.2: Comparison operators.

Comparison operator	Description
==	Outputs `True` if the two operands are equal.
!=	Outputs `True` if the two operands are not equal.
>	Outputs `True` if the left operand is greater than the right operand.
>=	Outputs `True` if the left operand is greater than or equal to the right operand.
<	Outputs `True` if the left operand is less than the right operand.
<=	Outputs `True` if the left operand is less than or equal to the right operand.

Table 2.6.3: Logical operators.

Logical operator	Description
`&`	Outputs `True` if the two operands are both `True`.
`|`	Outputs `True` if at least one of the operands is `True`.
`~`	Outputs the opposite truth value of the expression.

Subsetting data using relational operators.

Full screen

- Click the double right arrow icon to restart the kernel and run all cells.
- Examine the code and text below.
- In cell [7], change the == operator into the != operator. Restart the kernel and run all cells to see the changes.
- Click "File", then "Download as". Download the Jupyter notebook as a PDF or HTML file.

In [1]:
```
# Load the pandas package
import pandas as pd
```

In [2]:
```
# Load the country.csv data
country = pd.read_csv('country.csv')
```

In [3]:
```
# Display the country dataframe
country
```

Out[3]:

	Country	Continent	SurfaceArea	Population	Density	IndependenceDate	C
0	China	Asia	9572900	1277558000	133	1949-10-01	
1	Bangladesh	Asia	143998	129155000	897	1905-05-24	
2	Brazil	South America	8547403	170115000	20	1822-09-07	
3	India	Asia	3287263	1013662000	308	1905-04-30	
4	Norway	Europe	385207	5379000	14	1905-05-17	
5	United States	North America	9363520	278357000	30	1776-07-04	

In [4]:
```
# Select rows where the Continent is South America
country[country['Continent'] == 'South America']
```

Out[4]:

	Country	Continent	SurfaceArea	Population	Density	IndependenceDate	OfficialL
2	Brazil	South America	8547403	170115000	20	1822-09-07	Po

PARTICIPATION ACTIVITY 2.6.5: Subsetting data using comparison and logical operators.

1) What statement returns countries in Europe?

- `country[country['Continent'] = 'Europe']`
- `country[country['Continent'] == 'Europe']`
- `country['Continent'] == 'Europe'`

2) What statement returns countries with a population of at least 1,000,000?

- `country[country['Population'] > 1000000]`
- `country[country['Population'] >= '1000000']`
- `country[country['Population'] >=  1000000]`

3) What statement returns countries in Asia with a population greater than 250,000,000?

- `country[country['Continent'] == 'Asia']`
- `country[(country['Continent'] == 'Asia') & (country['Population'] > 250000000)]`
- `country[(country['Continent'] == 'Asia') | (country['Population'] >= 250000000)]`

CHALLENGE ACTIVITY 2.6.1: Subsetting dataframes using pandas.

537150.4174434.qx3zqy7

Start

This dataset contains information on taxi journeys during March 2019 in New York City. The data includes passengers, dis fare, tip, tolls, and total.

- Create a dataframe containing the passengers column of the taxisNY dataframe.

The code contains all imports, loads the dataset, and prints the passengers column.

main.py | **taxi.csv**

```
1 # Loads necessary packages
```

```
# Loads necessary packages
import pandas as pd

# Loads the taxi.csv dataset
taxisNY = pd.read_csv('taxi.csv')

# Subset a column of the taxisNY dataframe by specifying the column name
passengers = # Your code goes here

# Prints the column
print(passengers)
```


1 2 3 4 5

Check Next level

Other `DataFrame` methods

`pandas` can also perform other data science tasks, such as:

- Insert, update, and sort rows and columns.
- Identify and replace missing values.
- Identify and delete duplicate rows.

The table below lists common methods for dataframes. The table includes all required parameters and important optional parameters, but excludes infrequently used optional parameters. Additional methods are described elsewhere in this material and in the pandas API reference.

Table 2.6.4: Example dataframe methods.

Method	Parameters	Description

`drop()`	`labels=None`
	`axis=0`
	`inplace=False`
		Removes rows (`axis=0`) or columns (`axis=1`) from `dataframe`. `labels` specifies the labels of rows or columns to drop.

`drop_duplicates()`	`subset=None` `inplace=False`	Removes duplicate rows from `dataframe`. `subset` specifies the labels of columns used to identify duplicates. If `subset=None`, all columns are used.

`dropna()`	`axis=0`
	`how='any'`
	`subset=None`
	`inplace=False`
		Removes rows (`axis=0`) or columns (`axis=1`) containing missing values from `dataframe`. `subset` specifies labels on the opposite axis to consider for missing values. `how` indicates whether to drop the row or column if `any` or if `all` values are missing.

`insert()`	`loc`
	`column`
	`value`
		Inserts a column to `dataframe`. `loc` specifies the integer position of the new column. `column` specifies a string or numeric column label. `value` specifies column values as a Scalar or Series.

`replace()`	`to_replace=None`
	`value=`
	`  NoDefault.no_default`
	`inplace=False`
		Replaces `to_replace` values in `dataframe` with `value`. `to_replace` and `value` may be string, dictionary, list, regular expressions, or other data types.

`sort_values()`	`by`
	`axis=0`
	`ascending=True`
	`inplace=False`
		Sorts `dataframe` columns or rows. `by` specifies indexes or labels on which to sort. `axis` specifies whether to sort rows (0) or columns (1). `ascending` specifies whether to sort ascending or descending. `inplace` specifies whether to sort `dataframe` or return a new dataframe.

PARTICIPATION ACTIVITY 2.6.6: Dataframe methods.

Refer to the `country` dataframe:

	Country	Continent	Population
0	China	Asia	9572900
1	Bangladesh	Asia	143998
2	Brazil	South America	8547403
3	Norway	Europe	358207

Select the statement that returns the result shown in each question.

1)

	Country	Continent	Population
0	China	Asia	9572900
2	Brazil	South America	8547403
3	Norway	Europe	358207

- `country.dropna(axis=0, subset='Continent')`
- `country.drop_duplicates(subset='Continent')`
- `country.drop(labels='Continent', axis=1)`

2)

	Country	Continent	Population
1	Bangladesh	Asia	143998
3	Norway	Europe	358207
2	Brazil	South America	8547403
0	China	Asia	9572900

- `country.sort_values(axis=0, by='Population')`
- `country.sort_values(axis=1, by='Population')`
- `country.sort_values(axis=0, by='Population', ascending=False)`

2.7 matplotlib package

Learning goals

- Create line plots and scatter plots using `matplotlib`.
- Use `matplotlib` to add axis labels, title, and legend to plots.
- Use `matplotlib` to add style elements to plots.
- Use `matplotlib` to add grid lines to plots.

- Create multiple plots in one figure using `matplotlib`.

What is matplotlib?

Data visualization plays a big part in the data science lifecycle. In particular, data scientists often create data visualizations to gain insight on data and develop a scope for a machine learning task. While several data visualization packages are available in Python, `matplotlib` is the most common. ***matplotlib*** is a package used to create static, dynamic, and interactive plots. `Seaborn`, another common data visualization package used primarily for statistical graphs, is based on `matplotlib`.

`matplotlib` uses figures to hold plot elements, such as axes, labels, tick marks, and legend. Each element can be adjusted manually. Ex: Positions of minor and major tick marks can be controlled as well as the font sizes for the labels.

Figures can be created using `pyplot`. The `pyplot` library is a state-based interface to the `matplotlib` package that uses a similar syntax to MATLAB. Each line of code adds a plot element to the figure one at a time, while preserving previously added elements in the figure. To learn more about common functions within the `pyplot` library, see the pyplot documentation.

Table 2.7.1: Advantages of matplotlib.

Versatility	Can be used in Python scripts, shells, iPython, and Jupyter Notebooks
Familiarity	Uses MATLAB-style functions
Portability	Runs on MacOS, Linux, and Windows
Customizability	Can adjust details of a plot like axes, legends, and color

PARTICIPATION ACTIVITY 2.7.1: What is matplotlib?

1) The `matplotlib` package can ____.
   - create statistical models
   - display static and dynamic plots
   - compute a correlation coefficient

2) The `matplotlib` package works across multiple platforms.
   - True
   - False

3) For a figure created using the `matplotlib` package, _____ can be adjusted.

- only the axes labels
- only the axes tick marks
- both axes labels and tick marks

Plotting with matplotlib

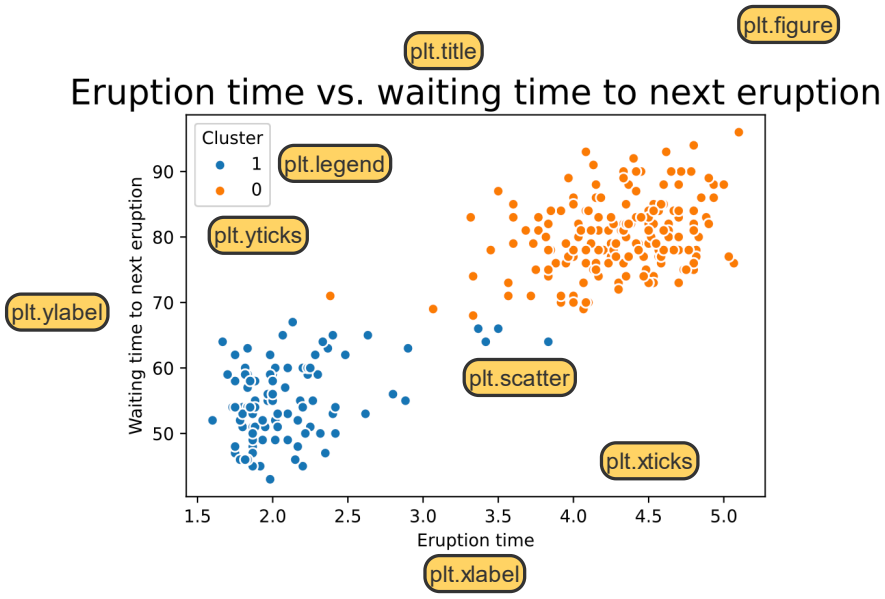
The first step to create a plot with `pyplot` is to import the library. Since `pyplot` is a library within the `matplotlib` package, dot notation is needed. Ex: `import matplotlib.pyplot as plt`.

The `pyplot` library can display different objects within a figure. The most common object is `plt.plot(x, y)`, where x and y are arrays of the same size, which creates a line plot connecting consecutive x- and y- coordinates. Another common object is `plt.scatter(x, y)`, which creates a scatter plot showing all pairs of x- and y- coordinates.

Other useful functions in the `pyplot` library are:

- `plt.figure()`—creates a new figure.
- `plt.show()`—displays the figure and all the objects the figure contains.
- `plt.savefig(fname)`—saves the figure in the current working directory with the filename `fname`.

PARTICIPATION ACTIVITY 2.7.2: Parts of a figure.



Animation content:

A scatter plot of eruptions and waiting times between eruptions are plotted. A matplotlib figure holds all plot elements: scatter plot, axes, title, axes labels, and legend. An axis is an object attached to a figure. Two axes are in the figure, horizontal and vertical. The horizontal axis is the eruption time and the vertical axis is the time between eruptions. The tick marks for the horizontal axis are set in intervals of 0.5 minutes, while the tick marks for the vertical axis are set in the intervals of 10

minutes, although these tick marks can be adjusted manually. Other objects like line plot, title, and legend can also be added.

Animation captions:

1. The figure object holds all plot elements.
2. An axis is an object attached to a figure. Tick marks and labels can be set manually.
3. Other objects like scatter plot, legend, and title can also be added.

Are the plt.figure() and plt.show() functions necessary?

`plt.figure()` is needed only when changing the default size of the figure. The size of the figure can be specified using the `figsize` parameter. Since a figure is implicitly created whenever a `plt.plot(x,y)` or `plt.scatter(x,y)` function is called, calling the `plt.figure()` function is not necessary if the default figure size is acceptable. Note: The default figure size is 6.4 inches by 4.8 inches.

Within Jupyter Notebooks or any interactive shell like IDLE, all elements within each figure are automatically displayed when cells or lines of code are run. However, the `plt.show()` function is necessary when `matplotlib` is used within a terminal or a script.

Creating plots.

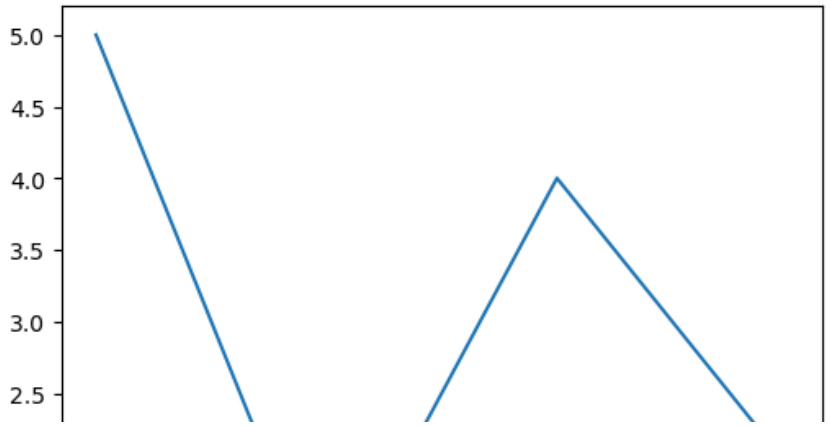
Full screen

- Click the double right arrow icon to restart the kernel and run all cells.
- Examine the code and text below.
- In cell [2], change the value of the `figsize` parameter from [6,5] to [9,6]. In cell [3], change `plt.plot(x, -4*x -4)` into `plt.plot(x, -3*x -2)`. Restart the kernel and run all cells to observe the changes.
- Click "File", then "Download as". Download the Jupyter notebook as a PDF or HTML file.

In [1]:
```
# Load matplotlib.pyplot, pandas, and numpy
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

In [2]:
```
# Create array for x- and y- coordinates
x = [1, 2, 3, 4]
y = [5, 1, 4, 2]
# Create a 6x5 figure
plt.figure(figsize=[6, 5])
# Create a line plot
plt.plot(x, y)
```

Out[2]: [<matplotlib.lines.Line2D at 0x11dc3d120>]



PARTICIPATION ACTIVITY 2.7.3: Plotting with matplotlib.

1) Which of the following lines of code creates a figure?

- `plt.figure`
- `plt.figure()`
- `plt.savefig(fname)`

2) Which of the following lines of code saves a figure using the filename photo.png?

- `plt.savefig(photo.png)`
- `plt.savefig('photo.png')`
- `photo.png.savefig()`

3) Under which circumstance should the `plt.figure()` be included in the code?

- When the default figure size does not need to be changed
- When the desired figure size is 6.4 centimeters by 4.8 centimeters
- When the desired figure size is 6.4 inches by 4.8 inches

Labeling plots

`matplotlib` has extensive support for adding titles, labeling axes, and adding text and annotation to different plots.

To add a title to a figure, the `plt.title()` function is used. Ex:
`plt.title('Alcohol-related fatalities in highways')`

The following functions are used to add text for the axes:

- `plt.xlabel()`: adds text for the x-axis
- `plt.ylabel()`: adds text for the y-axis

Adding mathematical expressions using LaTeX is also supported. Typesetting mathematical expressions using LaTeX can be done by enclosing an expression within a string with dollar signs. The letter `r` should precede a string so that a backslash is not treated as a Python escape. Ex:
`plt.title(r'Standard normal distribution $f(x) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}x^`

The following functions can be used to add text within the figure:

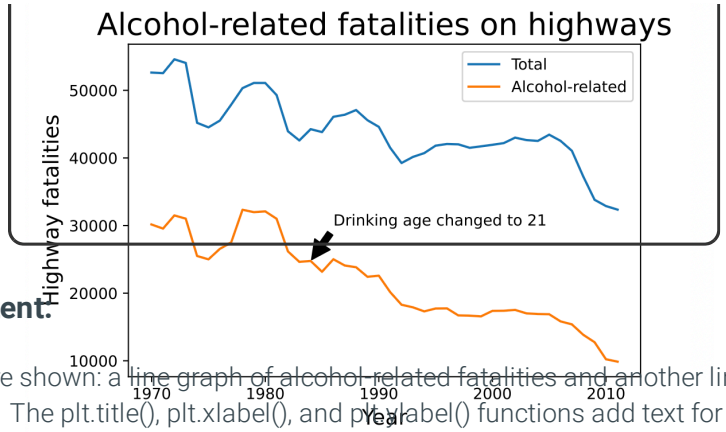
- `plt.text(x, y, s)`: adds string s to the figure at coordinates (x, y)
- `plt.annotate(s, xy, xytext)`: links string s at coordinates given by xytext to a point given by xy
- `plt.legend()`: adds legend in the figure

PARTICIPATION ACTIVITY 2.7.4: Labeling plots.

Python code

```
df = pd.read_csv('fatalities.csv')
plt.plot(df['Year'], df['Total'], label='Total')
plt.plot(df['Year'], df['Alcohol-related'],
         label='Alcohol-related')
plt.title('Alcohol-related fatalities on highways', fontsize=20)
plt.xlabel('Year', fontsize=14)
plt.ylabel('Highway fatalities', fontsize=14)
plt.annotate('Drinking age changed to 21',
             arrowprops=dict(facecolor='black', shrink=0.05),
             xy=(1984, 24762),
             xytext=(1986, 30000))
plt.legend()
```

Output



Animation content:

Two line graphs are shown: a line graph of alcohol-related fatalities and another line graph for total highway fatalities. The plt.title(), plt.xlabel(), and plt.ylabel() functions add text for the axes. The horizontal axis is the year and the vertical axis is the number of fatalities. The tick marks for the horizontal axis are set in intervals of 10 years, while the tick marks for the vertical axis are set in the intervals of 10,000 deaths, although these tick marks can be adjusted manually. The legend identifies the line graph for alcohol-related fatalities vs. total highway fatalities.

Animation captions:

1. The plt.plot() functions display line plots: The line plot on top displays total highway fatalities, and the line plot on the bottom displays alcohol-related fatalities.
2. The plt.title(), plt.xlabel(), and plt.ylabel() functions add text for the axes.
3. The plt.annotate() function adds text linked to specific coordinates within the figure. Here, the text is located at (1986, 30000) and the arrow points to (1984, 24762).
4. The plt.legend() function adds a legend within the figure, using the label from plt.plot(). matplotlib chooses the best location for the legend, but the location can be specified manually.

Labeling plots.

Full screen

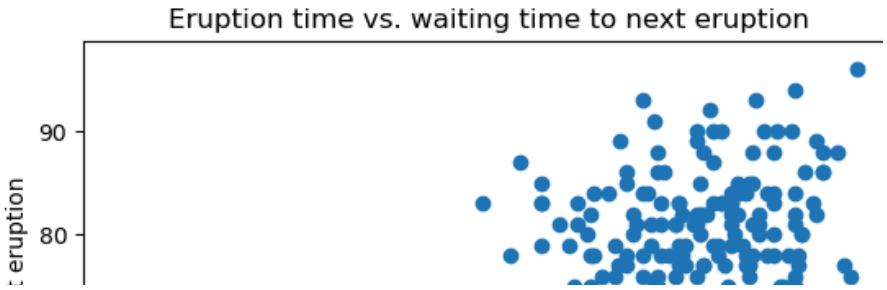
- Click the double right arrow icon to restart the kernel and run all cells.
- Examine the code and text below.
- Change the value of the `fontsize` parameter. Restart the kernel and run all cells to observe the changes.
- Click "File", then "Download as". Download the Jupyter notebook as a PDF or HTML file.

In [1]:
```
# Import packages
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```
# Load oldfaithful.csv data
df1 = pd.read_csv('oldfaithful.csv')
```


In [3]:
```
# Create a scatterplot
plt.scatter(df1['Eruption'], df1['Waiting'])
# Create label for x-axis
plt.xlabel('Eruption time')
# Create label for y-axis
plt.ylabel('Waiting time to next eruption')
# Create title
plt.title('Eruption time vs. waiting time to next eruption')
```

Out[3]: Text(0.5, 1.0, 'Eruption time vs. waiting time to next eruption')



PARTICIPATION ACTIVITY 2.7.5: Labeling plots.

1) Write a line of code that creates the title "Old Faithful eruptions" with a font size of 18.

```
plt.title(
                                    )
```

Check Show answer

2) Write a line of code that places the text "Scatter plot" at the coordinates (3, 4).


```
plt.text(
                                    )
```

Check Show answer

3) Write a line of code that places a legend at the lower right corner of the figure.

```
plt.legend(
)
```

Check Show answer

Styling plots

By default, lines and scatter plots are assigned certain colors and style. The style of different plots can be changed manually by adding a character or characters that correspond to color, line style, and marker style. Ex: `plt.plot(x, y, 'ro')` outputs a plot in red with circle markers. Style can also be defined using the parameters `color`, `linestyle`, and `marker`.

The table below gives a partial list of characters for color, line style, and marker style. If a marker style is not specified, the output will not show any markers.

Table 2.7.2: Characters for line color, line style, and marker style.

Character(s)	Line color/style	Character(s)	Marker style	Character(s)	Marker style
b	Blue	.	Point marker	1	Tri-down marker
g	Green	,	Pixel marker	2	Tri-up marker
r	Red	o	Circle marker	3	Tri-left marker
w	White	+	Plus marker	4	Tri-right marker
k	Black	X	X marker	h	Hexagon1 marker
y	Yellow	v	Triangle-down marker	H	Hexagon2 marker
m	Magenta	^	Triangle-up marker	D	Diamond marker
-	Solid line	<	Triangle-left marker	d	Thin diamond marker
:	Dotted line	>	Triangle-right marker	|	Vertical line marker
--	Dashed line	*	Star marker	_	Horizontal line marker
-.	Dashed-dot line	p	Pentagon marker	s	Square marker

Styling plots.

Full screen

- Click the double right arrow icon to restart the kernel and run all cells.
- Examine the code and text below.
- Change the value of the format strings in the plot commands. Restart the kernel and run all cells to observe the changes.
- Click "File", then "Download as". Download the Jupyter notebook as a PDF or HTML file.

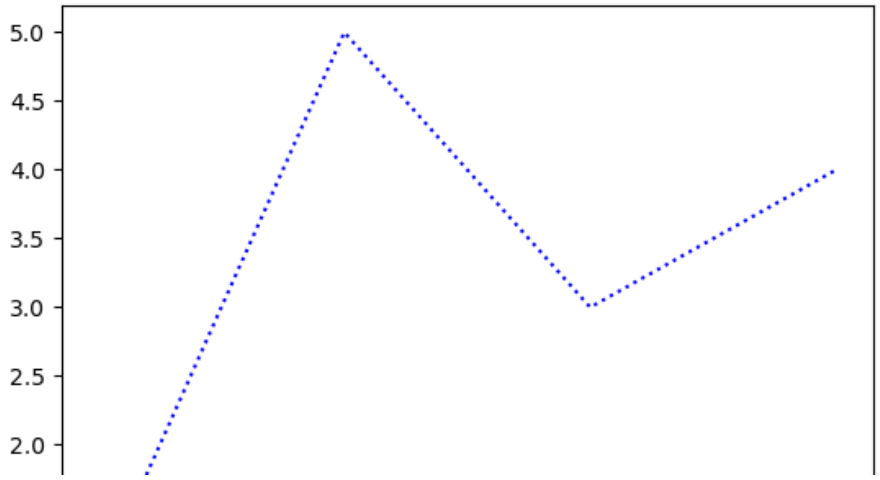
©zyBooks 03/17/24 16:51 2087217
Biniam abebe
UNTADTA5340OrhanSpring8wk22024

```
In [1]: # Load packages
        import matplotlib.pyplot as plt
        import pandas as pd
        import numpy as np
```

```
In [2]: # Create array for x- and y- coordinates
        x = [1, 2, 3, 4]
        y = [1, 5, 3, 4]
        # Create a line plot
        plt.plot(x, y, 'b:')
```



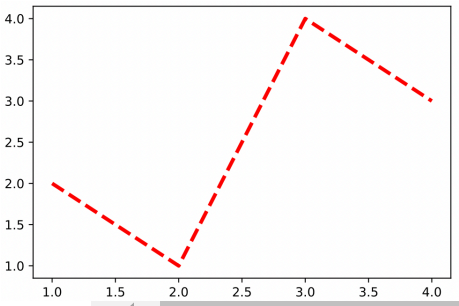
```
Out[2]: [<matplotlib.lines.Line2D at 0x138b28f10>]
```



PARTICIPATION ACTIVITY 2.7.6: Line style format strings.

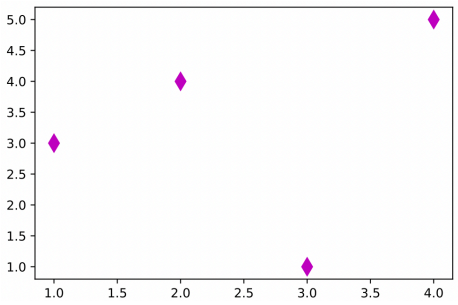
Select the format string used to style the line.

1)



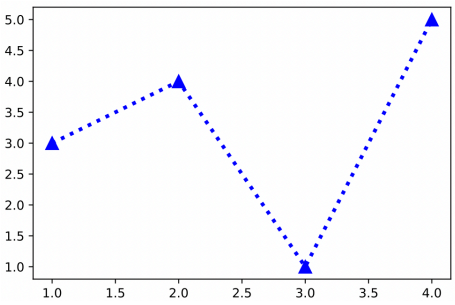
- `'r:'`
- `'rd'`
- `'r--'`

2)



- 'md'
- 'mD'
- 'm'

3)



- 'bt:'
- 'b^'
- 'b^:'

Grid lines

Grid lines are useful when determining the value of a point on a plot. The `plt.grid()` function adds grid lines to plots. By default, both x- and y- axes are displayed when using the `plt.grid()` function, but the axis parameter specifies which axis to display.

Properties such as color, linestyle, and linewidth can also be specified for the grid lines. Ex: `plt.grid(color='blue', linestyle='--', linewidth='5')`. Unlike line styles, characters for specifying grid lines cannot be combined.

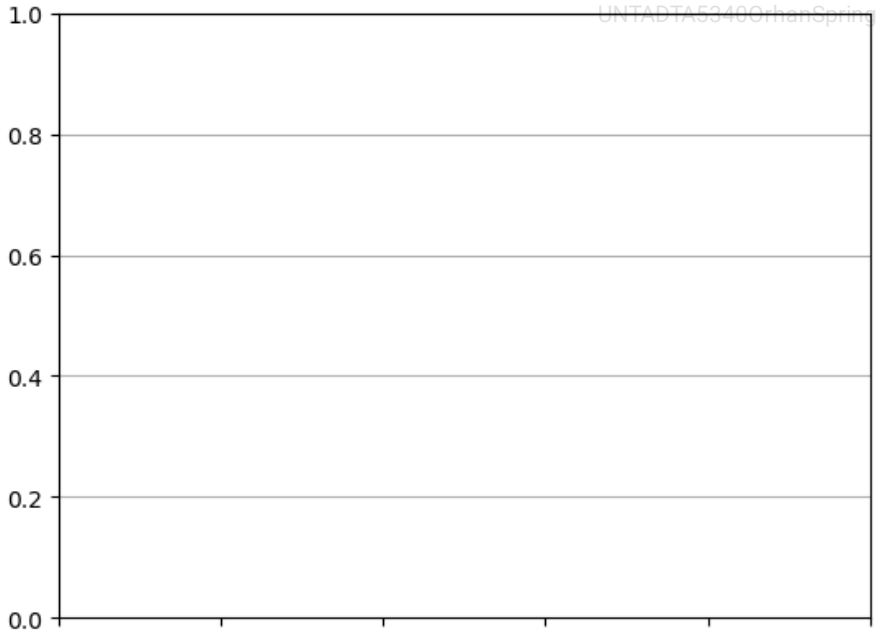
Adding grid lines.

Full screen

- Click the double right arrow icon to restart the kernel and run all cells.
- Examine the code and text below.
- In cell [3], change the linestyle parameter to ':'. Restart the kernel and run all cells to observe the changes.
- Click "File", then "Download as". Download the Jupyter notebook as a PDF or HTML file.

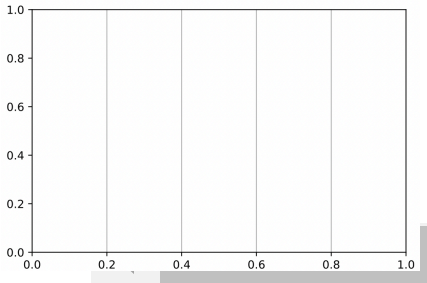
In [1]:
```
# Load packages
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```
# Create horizontal grid lines
plt.grid(axis='y')
```

PARTICIPATION ACTIVITY 2.7.7: Adding grid lines.

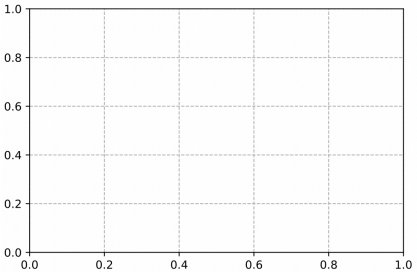
1) Write a line of code that creates vertical grid lines in the figure below.




```
plt.grid(
```
)

Check Show answer

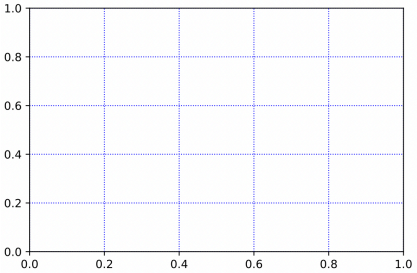
2) Write a line of code that creates the dashed vertical and horizontal grid lines below.




```
plt.grid(
```

Check Show answer

3) Write a line of code that creates the blue dotted vertical and horizontal grid lines below.




```
plt.grid(
```

Check Show answer

Multiple plots

Multiple plots can be drawn in the same figure using the `plt.subplot()` function. The `plt.subplot()` function takes three parameters: `nrows`, `ncols`, and `index`. The `nrows` and `ncols` parameters specify the figure layout, and the `index` determines which plot is created.

The function `plt.suptitle()` adds a title to the entire figure, not just the individual plots.

Multiple plots. Full screen

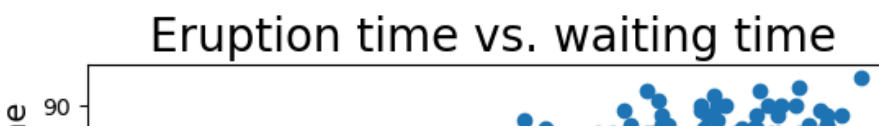
- Click the double right arrow icon to restart the kernel and run all cells.
- Examine the code and text below.
- Change the values of `nrows` and `ncols` to 1 and 3, respectively. Change the index of the second subplot to 3. Restart the kernel and run all cells to observe the changes.
- Click "File", then "Download as". Download the Jupyter notebook as a PDF or HTML file.

In [1]:
```
# Load packages
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```
# Load oldfaithfulCluster.csv data
df = pd.read_csv('oldfaithfulCluster.csv')
```

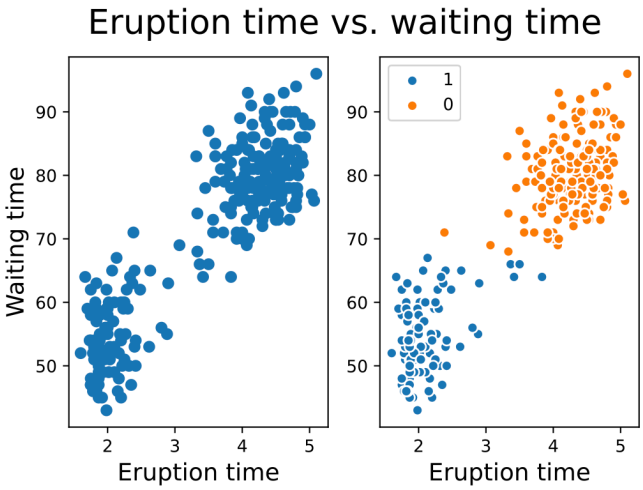

In [3]:
```
plt.subplot(2, 1, 1)
plt.scatter(df['Eruption'], df['Waiting'])
plt.title('Eruption time vs. waiting time', fontsize=20, c='black')
plt.ylabel('Waiting time', fontsize=14)

plt.subplot(2, 1, 2)
group1 = df[df['Cluster'] == 1]
group0 = df[df['Cluster'] == 0]
plt.scatter(group1['Eruption'], group1['Waiting'], label='1', edgeco
plt.scatter(group0['Eruption'], group0['Waiting'], label='0', edgeco
plt.xlabel('Eruption time', fontsize=14)
plt.ylabel('Waiting time', fontsize=14)
plt.legend()
```

Out[3]: <matplotlib.legend.Legend at 0x123728610>



PARTICIPATION ACTIVITY 2.7.8: Multiple plots.

Refer to the image below to answer the following questions.



1) Write the code to generate the title of the figure.

Check Show answer

2) Write the code for the subplot function that corresponds to the plot in the first row and second column of the figure.

Check Show answer

2.8 Case study: Hawks

Learning goals

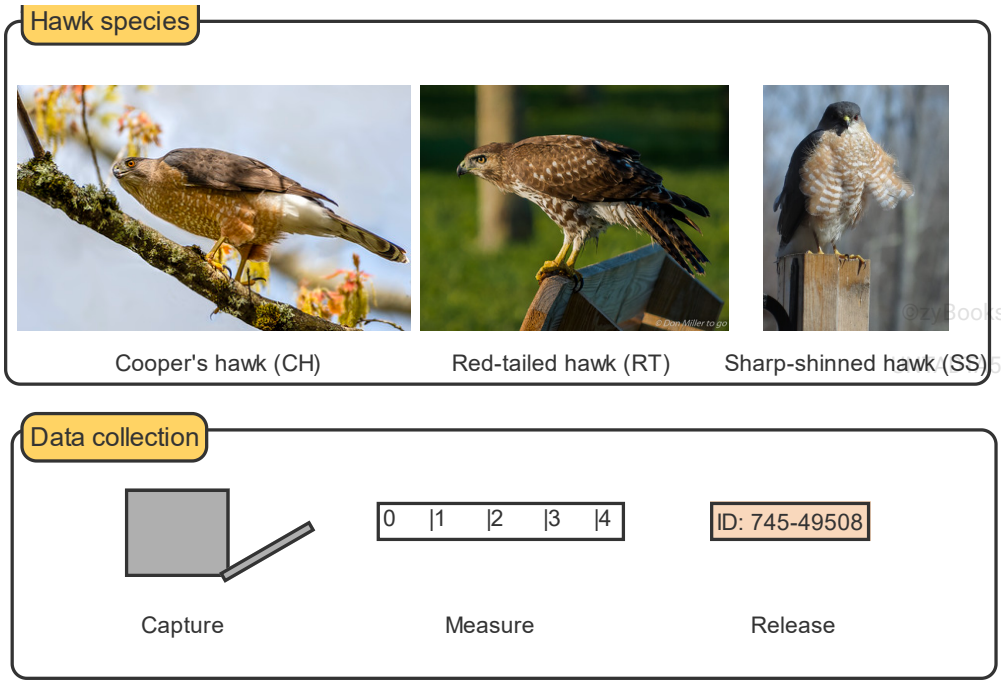
- Use `NumPy`, `pandas`, and `matplotlib` to explore a dataset.
- Connect data science packages to data science tasks.

Using data science packages to explore the hawks dataset

Longitudinal studies use repeated measurements of a population or feature over a period of time to track long-term changes. Longitudinal studies are used to evaluate medical treatments, monitor climate change, study animal populations, and model seasonal trends in customer behavior.

Construction of new housing developments and conversion of grasslands to farmlands are threats to hawk populations in Iowa. A group of student and faculty researchers used a 10-year longitudinal study to investigate long-term changes in the population of three local hawk species. Researchers collected data on body measurements, including age, sex, wing length, weight, bill length (culmen), and talon length (hallux). Researchers also used capture and release methods to track hawk movements over time.

PARTICIPATION ACTIVITY 2.8.1: The hawks dataset.



Animation content:

Static figure: Image of a Cooper's hawk, a red-tailed hawk, and a sharp-shinned hawk.

Animation captions:

1. Three hawk species are represented in the dataset: the Cooper's hawk (CH), red-tailed hawk (RT), and sharp-shinned hawk (SS).
2. Body measurements were taken using capture and release methods. Birds were captured using a net or trap.
3. After each bird was caught, body measurements were taken. Birds were sedated for comfort during measurement.
4. Before release, birds were tagged with a lightweight bracelet around their legs. The bracelet contained an ID number in case the bird was later recaptured.

PARTICIPATION ACTIVITY 2.8.2: Longitudinal data.

1) Longitudinal studies are conducted over an extended ____.

- ○ period of time
- ○ geographical area
- ○ number of subjects

©zyBooks 03/17/24 16:51 2087217
Biniam abebe
UNTADTA5340OrhanSpring8wk22024

2) Capture and release methods are used in wildlife studies because animals ____.

- are difficult to store and care for long-term
- can be easily recaptured
- grow and move over time

Applying NumPy to the hawks dataset

Many data scientists will begin an analysis by looking at the dataset. However, looking at a data table is not enough to gain meaningful insights. Data science packages like `NumPy`, `pandas`, and `matplotlib` are important tools when beginning a data science project.

`NumPy` provides functions for mathematical operations, like matrix algebra or calculating summary statistics. Data scientists may use `NumPy` for tasks like finding the dimensions of a dataset, reformatting features from a dataset, exploring features using mathematical operators, or calculating new features from a dataset.

NumPy functions for data science.

Full screen

The Python code below imports the hawks dataset and explores the data using functions from `NumPy`.

- Click the double right arrow icon to restart the kernel and run all cells.
- Examine the code below.
- Modify the code to explore different features.

```
In [1]:
import numpy as np
import pandas as pd

hawks = pd.read_csv('hawks.csv')
```

NumPy can be useful for finding the number of features (columns) and instances (rows) in a


```
In [2]:
hawks.shape

Out[2]: (891, 7)

In [3]:
hawks.size

Out[3]: 6237
```

Some data science models require features to be formatted as an array.

```
In [4]:
np.ravel(hawks['Wing'])

Out[4]: array([385. , 381. , 265. , 205. , 412. , 370. , 375. , 412. , 405. ,
               393. , 371. , 390. , 416. , 436. , 418. , 381. , 378. , 396. ,
               399. , 416. , 415. , 392. , 380. , 399. , 401. , 205. , 427. ,
               395. , 362. , 396. , 391. , 413. , 371. , 385. , 378. , 416. ,
               193. , 171. , 233. , 384. , 382. , 390. , 390. , 393. , 378. ,
               398. , 412. , 400. , 422. , 202. , 394. , 369. , 252. , 240. ,
```

Data source: Ann Cannon, George Cobb, Bradley Hartlaub, Julie Legler, Robin Lock, Thomas Moore, Allan Rossman, and Jeffrey Witmer. "Stat2Data: Datasets for Stat2", 2019. R package version 2.0.0. https://CRAN.R-project.org/package=Stat2Data

PARTICIPATION ACTIVITY 2.8.3: Exploring the hawks dataset with NumPy.

1) How many instances are in the hawks dataset?

- 7
- 891
- 6,237

2) What is the smallest wing length of a hawk in the data?

- 37.2 mm
- 57 mm
- 385 mm

3) How many hawks have a wing length less than 100 mm?

- 0
- 1
- 4

4) Calculate the average weight of all hawks in the dataset.

- 315.9 gm
- 771.6 gm
- 970 gm

Applying pandas to the hawks dataset

`pandas` contains functions for storing and manipulating datasets. Data scientists may use `pandas` to import and export datasets from Python, subset datasets, or insert new features into a dataset.

pandas functions for data science. Full screen

The Python code below imports the hawks dataset and explores the data using functions from `pandas`.

- Click the double right arrow icon to restart the kernel and run all cells.
- Examine the code below.
- Modify the code to explore different features.

In [1]:
```
import pandas as pd

hawks = pd.read_csv('hawks.csv')
hawks
```

Out[1]:

	Species	Age	Wing	Weight	Culmen	Hallux	Tail
0	RT	I	385.0	920	25.7	30.1	219
1	RT	I	381.0	990	26.7	31.3	235
2	CH	I	265.0	470	18.7	23.5	220
3	SS	I	205.0	170	12.5	14.3	157
4	RT	I	412.0	1090	28.5	32.2	230
...	...	...	...	...	...	...	...
886	RT	I	380.0	1525	26.0	27.6	224
887	SS	I	190.0	175	12.7	15.4	150
888	RT	I	360.0	790	21.9	27.6	211
889	RT	I	369.0	860	25.2	28.0	207
890	RT	A	199.0	1290	28.7	32.1	222

891 rows × 7 columns

Use `pandas` to create a subset of the dataset for each species.

In [2]:
```
CoopersHawk = hawks[hawks['Species'] == 'CH']
RedTailed = hawks[hawks['Species'] == 'RT']
```

PARTICIPATION ACTIVITY 2.8.4: Exploring the hawks dataset with pandas.

1) How many red-tailed hawks are in the dataset?
   - 69
   - 255
   - 567

2) What is the tail length of the hawk in row 31?
   - 31.3
   - 219
   - 238

3) Is the following statement true or false?

```
hawks.iloc[29, 3] >= 1000
```

   - True
   - False