ontinue : skips to the next iteration of the loop

reak : Stops executing the loop entirely

f-strings : string interpolation

* Use :

if : check a single condition

if/else : zig or zag, but no other option

if/elif/elif/else : more than 2 outcomes.

Functions

```
def increment(n):
    return n + 1
```

- the keyword : def
- Name of the function : increment
- Set of parentheses : Define the inputs to function
- Body of the function : Everything indented after the first line
- Return Statement : Body of the function inside

rgument :
alue a function
is called with.

arameter :
lace holder
or an argument.

> * whatever expression follows the return keyword will be the output of the defined function

Take on the value of the functions arguments each time it is called.

Default Values : Allows to either specify the argument or leave it out when the function is called.

```
def sayhello(name = 'World', greeting = 'Hello'):
    return '{} {}!'.format(greeting, name)
```

IN : sayhello()
UT : 'Hello, World!'

IN : sayhello('Codeup') IN : sayhello('Codeup', 'Salutations')
'Hello, Codeup!' OUT : Salutations, Codeup!

Keyword arguments : Specify arguments by their name.

```
sayhello(greeting = 'Salutations', name = 'Codeup')
```

ut : 'Salutations, Codeup!'

* Key word arguments MUST come after any posistional arguments.

IN: sayhello('Codeup', greetings = 'Salutations') # Okay

OUT: 'Salutations, Codeup!'

IN: sayhello(greeting = 'Salutations', 'Codeup') # Error

Imports

- 3 places to import from:
  1) Python Standard Library
     - Comes w/ the Python language
  2) 3rd Party Libraries
     - Require installation: conda or pip
  3) Our own code!
     Break our code down into seperate files + use code from one file in another

ex) Simple import

```
IN: import time   ← import keyword + module name

# After importing, you can reference variables or functions
  defined in the module using a (.) after the module name

IN: time.sleep(5)
    print('All Done!')
```

```
# Rename module
IN: import time as t

t.sleep(5)
print('All Done!')
```

```
# import individual functions or variables
  using from and import
IN: from time import sleep

sleep(5)
Print('All Done!')
```

Importing From my own CODE

1) We have util.py

```
def sayhello():
    print("Hello, World!")
```

→ 2) Import file

```
import util
util.sayhello()
```

* Only have definitions inside of a module *

Scatter Charts

- Require two data series to plot against each other.

```
# Scatter chart ex)
# plotting random data
x1 = [1,2,3,4,5,6,7,8]
x2 = [5,6,7,8,9,10,11,12]
plt.scatter(range(len(x1)), x1, s=50, c='red')
plt.scatter(range(len(x2)), x2, s=30, c='green')
plt.show()
```

Labels / Titles / Axes

Title: `.title`

Axis Labels:
- x-axis: `.xlabel`
- y-axis: `.ylabel`

Custom Ticks
- `.xticks`
- `.yticks`

- `.rotation` - Rotate axis ticks
- `.annotate` Annotate chart
- `.legend` Add legend
- `.savefig` - Save Figures (png default)

Custom Line Types
- `:` dotted line
- `--` dashed line

Adding Text
- `.text` - adds text
  - Specify: color, fontsize

Figures (Size + Legend)
- Change properties of the figure w/: `plt.figure`
  - * Must be done BEFORE adding to the chart

Subplots

- Creating multiple charts together
- Must specify:
  - Total # of rows
  - Total # of columns
  - which chart is the 'Active' chart
- `.suptitle` - Specifies an overall title for the chart

Subplot syntax example:

```
n_rows = 1
n_cols = 2

# Data
x = [1,2,3,4,5]
y = [5,4,3,2,1]
z = [1,2,3,4,5]

# Plot first subplot
plt.subplot(n_rows, n_cols, 1)
plt.plot(x, y)
plt.title('x ~ y')

# Second subplot
plt.subplot(nrows, n_col, 2)
plt.plot(x, z)
plt.title('x ~ z')

plt.show()
```

Histograms - Create using .hist function s

grouping data

```
x = [1,1,2,3,3,3,4,4,5]
plt.hist(x)
plt.show()
```

```
# histogram with "bins" specified
x = [1,1,2,3,3,3,4,4,5]
plt.hist(x, bins = [0,1,2,3,4,5,6])
plt.show()
```

align

Tells matplotlib how the bins should be aligned:

left, middle, right

ex) `plt.title('align = "right"')`

```
# Plot multiple histograms together * Useful to set alpha for the colors
x1 = [randint(1,5) for _ in range(20)]
x2 = [randint(1,5) for _ in range(20)]

plt.hist(x1, bins=[0,1,2,3,4,5,6], align='left', edgecolor='black',
         alpha=0.5, color='blue')
plt.hist(x2, bins=[0,1,2,3,4,5,6], align='left', edgecolor='black',
         alpha 0.5, color='blue')
plt.show()
```

```
# Plot Histogram Side x Side
```

* must pass the data series as a tuple to the hist function *

```
x1=[randint(1,5) for _ in range(20)]
x2=[randint(1,5) for _ in range(20)]

plt.hist((x1,x2), bins=[1,2,3,4,5,6], align='left', edgecolor=
    'black', alpha=0.6, label=['$x_1$ $x_2$'])

plt.legend()
plt.show()
```

Numpy

Library for representing + working w/ large + multi-dimensional arrays (vectors)

- Most libraries depend on numpy

```
import numpy as np
```

Indexing:

```
# Create numpy array (by passing a list)
IN:  a = np.array([1,2,3])
     a
OUT: array([1, 2, 3])

# Multi Dimensional Array (by passing lists of lists)
IN:  matrix = np.array([[1,2,3],
                        [4 5 6]
                        [7 8 9]])
     matrix

# Obtain the element at the second column in the second row
IN: matrix[1,1]

# First 2 elements of Last 2 rows
IN: matrix[1:, :2]

# Index with boolean sequence
IN: should_include_elements = [True, False, True]
    a[should_include_elements]
```

- Arrays of Booleans = Beating Heart of Filtering/Transforming Arrays
- Boolean Masking/Filtering

Vectorized Operation - Operations are automatically applied (like behavior for arrays w/o loops) to every element in the vector

To array, add 1

N: original_array = np.array([1,2,3,4,5])
original_array + 1

Get all positive numbers in my_array

N: my_array[my_array > 0]

Array Creation

- np.random.randn
  ↓
  np.random.randn(10)
  - Creates an array of specified length of random numbers drawn from the Standard distribution
  - Can also pass second argument to define the shape of a two dimensional array
    (N: np.random.randn(3,4)

Draw from a normal distribution with mean (μ) and standard deviation (σ)

N:
mu = 100
sigma = 30

→ 1) Multiply by the standard deviation
2) Add the mean

sigma * np.random.randn(20) + mu

np.zeros
np.ones - Provides ability to create arrays of a specified size full of either 0s or 1s

np.full - Allows to create an array of the specified size w/ a default value

Zeros, Ones, Full
print(
print(
print(

Pandas - Python library for representing dataframes

Series: 1-Dimensional representations of data

Data frames: Basic concepts and manipulation of pandas 2-D data structures.

Advanced Dataframes - More advanced data frame manipulation

Pandas Series: import pandas as pd

Creating Series

```
series = pd.Series([100, 43, 26, 17])
type(series)
```

Series Properties

index - way to reference items in the series

type - Data type of the elements in the series.
- int - integer (whole #)
- bool : true/false values
- float - decimals
- object: strings
- category - Fixed set of string values

name - Optional, human friendly name for series

.astype (Convert between data types)

```
N: string_series = pd.Series([3.5 4.5 6]).astype('str')
   string_series
```

any - Check if any value is true (series < 0).any

all - Check if All values in the series are true (series > 0).all()

head()/.tail() - Look at first and last several values in a series.

value_counts - Gives count of unique values in a series

isin - Check if each value is in a set of values

apply() - Appy a function to a series

Plotting (Pandas Series)

import pandas as pd

In: %matplotlib inline
import matplotlib.pyplot as plt

series.plot() or can specify: series.plot.hist()

Dataframes

df. Represent tabular, 2-Dimensional data

.info - prints useful information from the dataframe
.describe - gives summary of the numerical values in df.

DF Attributes

- dtypes: Data type in each column.
- shapes: Number of rows & columns in the df
- columns: List of column names
- index: Labels for each row

#Subsetting Dataframes

- #Multiple columns - df[['name', 'math']]

```
columns = ['name', 'math']
df[columns]
```

#Individual Columns (Each column is a series)

Way 1: df.math Way 2: df['math']

#Accessing Row Subsets

- .head: first n (default 5) rows
- .tail: last n (default 5) rows
- .sample: for a sample of rows

Dropping/Renaming Columns

.drop(ex) - df.drop(columns=['english', 'reading'])

.rename(ex) - df.rename(columns={'name': 'student'})

```
# Chain Dataframes
df.drop(columns=['english', 'reading']).rename(columns
    {'name': 'student'})
```

= Creating New Columns

) Boolean value creation: df.math >= 70

) Add coulmn to dataframe:

```
df['passing_math'] = df.math > 70
```

OR USE .assign method:

```
df.assign(passing_english = df.english >= 70)
```

Sorting Dataframes

sort_values: df.sort_values(by='english')

Desc. Order Sort:

```
df.sort_values(by='english', ascending=False)
```

```
df[df.english > 90].sort_values(by='english').head(1).name
```

Breakdown of expression.

) df: Initial variable holds dataframe

)[df.english > 90]: Dataframe is subset to find grades over 90

). sort_values(by='english'): Remaining rows sorted by english

). head(1): Take the first record

) .name: extract the name part of the record.

Advanced Dataframes:

Create Dataframes From Lists + Dictionaries

Dictionary: `pd.DataFrame({'A': [1,2,3], 'B': [4,5,6]})`

List: `pd.DataFrame([[1 2 3], [4 5 6]])`

Numpy Array:

```
data = np.array([[1 2 3],[4 5 6]])
pd.DataFrame(data, columns = ['a', 'b', 'c'])
```

	A	B
0	1	4
1	2	5
2	3	6

Dictionary output

	0	1	2
0	1	2	3
1	4	5	6

List output

	a	b	c
0	1	2	3
1	4	5	6

Array Output

Create from Text Files

- Most common: read_csv
  - Others: read_table, read_json

Create from SQL

create an "env.py" file to the gitignor

Host

w/ SQL Username

Password

Pandas x SQL Example

```
from env import host, user, password

url = f'mysql+pymysql://{user}:{password}@{host}/employees'
```

Aggregation (together)

- agg - specifies a way to aggregate a series of numerical values

```
# Aggregate - df.reading.agg('min')

# More Detailed Aggregation
df[['english' 'reading' 'math']].agg(['mean' 'min' 'max'])
```

Grouping:

* use .transform to add on to existing dataframe

- groupby - used to create a grouped object that can an apply an aggregation on

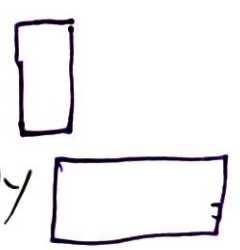
```
# ex) Find highest math grade from each classroom
df.groupby('classroom').math.max()

#ex) Groupby / Aggregate for Multiple Aggregations
df.groupby("classroom").math.agg(['min', 'mean', max])
```

Merging And Joining:

pd.concat - Combine dataframes vertically

Pd.merge - Combine dataframes horizontally



```
# Concat
df1 = pd.DataFrame({'a': [1, 2, 3]})
df2 = pd.DataFrame({'a': [4, 5, 6]})

# Merge
pd.merge(users, roles, left_on = 'role_id', right_on = 'id',
         how = 'left')
```

Seaborn - Builds on matplotlib

```
import seaborn as sns

# tips (Info about restaurants customers/total bill/tip)
tips = sns.load_dataset('tips')
```

Scatter Charts

replot: Function needed to supply keyword arguments to

- data - Pandas dataframe, that contains data to plot
  - x - Name of the x value column
  - y - Name of the column that will be y value

- col - Specify the name of the column

IN: `sns.replot(x = 'total_bill', y = 'tip', col = 'time', data = tips)`

- style - Change shape of the data points.
- hue - Seperate data points by color

N: `sns.replot(x = 'total_bill' y = 'tip', hue = 'smoker', data = tip`

↗ these can be combined as well

Other Chart Types

- distplot - Used to check the distribution of a dataset
- boxplot - Depict groups of numerical data through their quartile
- heatmap - Visualizes tabular data
- pairplot - Visualize relationship between every quantitive featur in a dataset.
- jointplot - Combine two visualizations