Module 3

*Command Line*

chdir – find the current directory with command

dir – displays list of folders and files

cd.. – go back one level

cd <location> - add a new level to the path

cd..\Desktop – take you to desktop folder, which is back one directory; “\” is a path separator; “Desktop” is included after backslash

If you know folder structure, then type “cd C:\Users\KyleKato\Desktop

Text

Description automatically generated

*Git Bash*

Navigating thru Git Bash is the same as command prompt

Pwd – print working directory; current location

Similar to using dir in cmd

Ls – list directory

Can use 3 different methods to reach interpret Python code:

Command Prompt – type python

Git Bash – type python

Python Interpreter

To launch VS Code from cmd, type – ‘code’

Open cmd and type in ‘python’

type() – function used to determine data type; type(“<variable>”)

>>> type(3)

<class 'int'>

>>> ballots = 1,341

>>> ballots

(1, 341)

>>>

>>> type(ballots)

<class 'tuple'>

>>>

Boolean – true or false; type(True): ‘bool’

|  |  |
| --- | --- |
| **Data Type** | **Python Classification** |
| Integers | <class 'int'> |
| Float point numbers | <class 'float'> |
| Strings | <class 'str'> |
| Boolean | <class 'bool'> |

Declared Variables:

num\_candidates = 3

winning\_percentage = 73.81

candidate = "Diane"

won\_election = True

Keywords: AKA reserve words that have special language functionality. No variable can have the same name as keyword.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **False** | **def** | **if** | **raise** | |
| None | del | import | return |
| True | elif | in | try |
| and | else | is | while |
| as | except | lambda | with |
| assert | finally | nonlocal | yield |
| break | for | not |  |
| class | from | or |  |
| continue | global | pass |  |

Can also get a list if you type ‘help(“keywords”)’

Arithmetic Operators:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Use** |
| + | Adds two numbers. | *x* + *y* |
| – | Subtracts one number from another. | *x* – *y* |
| \* | Multiplies two numbers. | *x* \* *y* |
| / | Divides one number by another. This always results in a floating-point decimal number. | *x* / *y* |
| % | The “%” is known as the **modulus**. When used in place of “/” it will divide one number by another, and return the remainder of the division. | *x* %*y*  (remainder of *x*/*y*) |
| // | Divides one number by another and returns an integer. This is known as **floor division**. | *x* //*y* |
| \*\* | Raises a number to a power. | *x*\*\**y*  (*x* to the power *y*) |

List is an array that contains multiple data items, like list of countries

**Negative indexes** are used to identify a list item's position relative to the end of the list.

>>> print(counties[-1])

Jefferson

Three properties of lists:

1. Use **indexing** and **slicing** to retrieve specific items from list

2. We can add or remove items from a list, which makes lists a **dynamic** data structure

3. We can change the contents in a list. For example, we can change "Jefferson" to "El Paso." This means lists are **mutable:** we can change one or more items in a list to something else.

>>> counties = ["Arapahoe","Denver","Jefferson"]

When we want to add items to a list but the list has not been declared, we must first declare an empty list. An **empty list** can be declared with the following syntax:

 my\_list = [ ]

Alternatively, you can use the built-in function list() to create an empty list:

my\_list = list()

An index of a variable is its position in the array. Here are some general rules for indexing:

1. Each item in a list has an index that specifies its position in the list.

2. Indexing starts at 0. Therefore, the index of the first item is 0, the index of the second number is 1, and so on.

3. Because indexing begins at 0, the index of the last item in a list is 1 less than the number of items in the list.

First Item in the list:

>>> counties[0]

'Arapahoe'

Negative Index - identify a list item's position relative to the end of the list

>>> print(counties[-1])

Jefferson

To get the **second-to-last** item in a list, we would type counties[-2], and so forth.

Len( ) – total number of items in a list

>>> len(counties)

3

**Slicing** is used to get specific items from a list; Use this expression:

List [start : end]

Return the first and second items in the list:

>>> counties[0:2]

['Arapahoe', 'Denver']

Items can be added to an empty list or a list that already exists by using: append ( ) with syntax: list.append ( )

>>> counties.append("El Paso")

>>> counties

['Arapahoe', 'Denver', 'Jefferson', 'El Paso']

To specify where in a list to add a new item, select the location with an index by using the following syntax:

List.insert (index , obj)

>>> counties.insert(2, "El Paso")

>>> counties

['Arapahoe', 'Denver', 'El Paso', 'Jefferson', 'El Paso']

To remove an instance from a list:

>>> counties.remove("El Paso")

>>> counties

['Arapahoe', 'Denver', 'Jefferson', 'El Paso']

>>>

Can also remove from a list using: pop ( )

>>> counties.pop(3)

'El Paso'

>>> counties

['Arapahoe', 'Denver', 'Jefferson']

>>>

Pop only uses the indexing; Use remove if identifying string

We can change the elements inside a list. To change a list use:

list [ index ]

Change Jefferson county to El Paso:

>>> counties[2] = "El Paso"

>>> counties

['Arapahoe', 'Denver', 'El Paso']

>>>

**Tuples** are similar to lists in Python, with a major exception: once you create a tuple, it cannot be changed

Empty tuple:

my\_tuple = ( )

or built-in method

my\_tuple = tuple ( )

counties\_tuple = ("Arapahoe","Denver","Jefferson")

>>> len(counties\_tuple)

3

>>> counties\_tuple[1]

'Denver'

A **dictionary** is an object that stores a collection of data

A Python dictionary has a **key** and a **value**, or **key-value** **pairs**. Very similar to a dictionary that contains definitions

Timeline

Description automatically generated

{key:value}

Or

{key1:value1, key2:value2}

There are two key rules for dictionaries:

1. Values in a dictionary can be objects of any type: integers, floating-point decimals, strings, Boolean values, datetime values, and lists.

2. Keys must be immutable objects, like integers, floating-point decimals, or strings. Keys cannot be lists or any other type of mutable object.

To initialize or create an empty dictionary, we use the following syntax:

my\_dictionary = {}

Or you can create a dictionary with the built-in Python

my\_dictionary = dict()

Number of voter for Aphrodite

>>> counties\_dict = {}

>>> counties\_dict["Arapahoe"] = 422829

>>> counties\_dict

{'Arapahoe': 422829}

>>> counties\_dict["Denver"] = 463353

>>> counties\_dict["Jefferson"] = 432438

>>> counties\_dict

{'Arapahoe': 422829, 'Denver': 463353, 'Jefferson': 432438}

Length of the dictionary:

Number of items in the dictionary

len()

>>> len(counties\_dict)

3

>>>

Get All Values and Keys

To get all the keys and values printed to the screen, simply print the dictionary name; Use:

items()

>>> counties\_dict.items()

dict\_items([('Arapahoe', 422829), ('Denver', 463353), ('Jefferson', 432438)])

>>>

Information inside the below is what is known as view object

dict\_items([])

You cannot use list indexing with the items() method.

Get All Keys

>>> counties\_dict.keys()

dict\_keys(['Arapahoe', 'Denver', 'Jefferson'])

>>>

Get All Values

To retrieve only the values from a dictionary, add the values() method

>>> counties\_dict.values()

dict\_values([422829, 463353, 432438])

>>>

**Just like the**items() **and** keys()**methods, the**values()**method will return a view object that contains the values of the dictionary as a list**

Get a Specific Value

There are 2 methods that can be used to get a specific value from a dict

Pass the key inside the parentheses to “get” the value

get()

Let's "get" the value or the number of registered voters, in Denver County

>>> counties\_dict.get("Denver")

463353

Get the number of registered voters in Arapahoe County, we can wrap the key in brackets:  
dictionary\_name[key]

>>> counties\_dict['Arapahoe']

422829

>>> counties\_dict["Arapahoe"]

422829

Lists of Dictionaries

List of dictionaries – dictionary wrapped in brackets {}

[{key1:value1, key2:value2}, {key1:value3, key2:value4}]

Create a list voting.data then add, or append, each dictionary to the list

>>> voting\_data = []

>>> voting\_data.append({"county":"Arapahoe", "registered\_voters": 422829})

>>> voting\_data.append({"county":"Denver", "registered\_voters": 463353})

>>> voting\_data.append({"county":"Jefferson", "registered\_voters": 432438})

>>> voting\_data

[{'county': 'Arapahoe', 'registered\_voters': 422829}, {'county': 'Denver', 'registered\_voters': 463353}, {'county': 'Jefferson', 'registered\_voters': 432438}

If you need to create an algorithm that calculates the percentage of votes a candidate receives in an election, you might write a simple algorithm like this:

# How many votes did you get?

my\_votes = int(input("How many votes did you get in the election? "))

# Total votes in the election

total\_votes = int(input("What is the total votes in the election? "))

# Calculate the percentage of votes you received.

percentage\_votes = (my\_votes / total\_votes) \* 100

print("I received " + str(percentage\_votes)+"% of the total votes.")

If Statements

Diagram

Description automatically generated

In Python, the general format for the if statement is to write a single alternative decision as follows:

if condition:

statement 1

statement 2

If statements says if certain lines of code should run, the if statement check if true and if ture runs the block of code

If statements begin with the word ‘if’ followed by a condition

1. Can be expression like:
   1. if len(counties) > 2:
   2. At the end of the condition is a colon
2. An indention is required after the if for the python interpreter
3. counties = ["Arapahoe","Denver","Jefferson"]
4. if counties[1] == 'Denver':
5. print(counties[1])

If-else:

temperature = int(input("What is the temperature outside? "))

if temperature > 80:

print("Turn on the AC.")

else:

print("Open the windows.")

Nested If-Else

Chart, radar chart

Description automatically generated

#What is the score?

score = int(input("What is your test score? "))

# Determine the grade.

if score >= 90:

print('Your grade is an A.')

else:

if score >= 80:

print('Your grade is a B.')

else:

if score >= 70:

print('Your grade is a C.')

else:

if score >= 60:

print('Your grade is a D.')

else:

print('Your grade is an F.')

Or you can use ElIF

# What is the score?

score = int(input("What is your test score? "))

# Determine the grade.

if score >= 90:

print('Your grade is an A.')

elif score >= 80:

print('Your grade is a B.')

elif score >= 70:

print('Your grade is a C.')

elif score >= 60:

print('Your grade is a D.')

else:

print('Your grade is an F.')

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Example** |
| in | Returns True if a sequence with the specified value is present in the object. | counties = ["Arapahoe","Denver","Jefferson"] if "Arapahoe" in counties: print("True") else: print("False")  This prints "True" because *Arapahoe*is in the counties list. |
| not in | Returns True if a sequence with the specified value is**not**present in the object. | counties = ["Arapahoe","Denver","Jefferson"] if "El Paso" not in counties: print("True") else: print("False")  This prints "True" because *El Paso*is not in the counties list. |

Membership Operators:nkl,njklnm

counties = ["Arapahoe","Denver","Jefferson"]

if "El Paso" in counties:

print("El Paso is in the list of counties.")

else:

print("El Paso is not the list of counties.")

Output:

El Paso is not the list of countries

Logical Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Example** |
| and | Evaluates two Boolean expressions into one compound expression. The compound expression is true if **both** Boolean expressions are true.  If one of the expressions is false, then the compound expression is false. | x = 5 y = 5 if x == 5 and y == 5: print("True") else: print("False")  This prints "True" because *x* = 5 is true and *y* = 5 is true. |
| or | Evaluates two Boolean expressions into one compound expression. The compound expression is true if **either** Boolean expression is true.  If one of the expressions is false, then the compound expression is true. If both expressions are false, then the compound expression is false. | x = 5 y = 5 if x == 3 or y == 5: print("True") else: print("False")  This prints "True" because *x* = 3 is false and *y* = 5 is true. |
| not | Evaluates a Boolean expression. The expression is true if the conditional is **false**. | x = 5 y = 5 if not(x > y): print("True") else: print("False")  This prints "True" because *x* is not greater than *y*. If *x* = 6, then it would print "False" because *x* is greater than *y*. |

Use AND operator

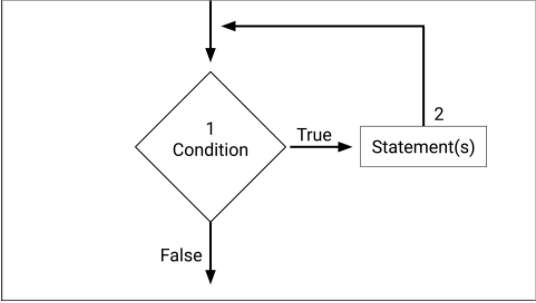
if "Arapahoe" in counties and "El Paso" in counties:

print("Arapahoe and El Paso are in the list of counties.")

else:

print("Arapahoe or El Paso is not in the list of counties.")

1. Condition-controlled loop uses a true / false condition to control the number of times that it repeats:
   1. While
2. Count-controlled loop repeats a specific number of times depending on conditions
   1. For
3. While loops – test if a conditional is true; If true then code will preform tasks
   1. A condition that is tested for a true or false value
   2. A statement or statements that are repeated as long as the condition is true



Using the while loop Syntax:

x = 0

while x <= 5:

    print(x)

    x = x + 1

For Loops:

Iterate, or run through, a program specific number of times before it stops

A For loop can be written like if and if-else

for item in [items]:

    statement 1

    statement 2

For Loop using Counties list

counties = ["Arapahoe", "Denver", "Jefferson"]

for county in counties:

    print(county)

Arapahoe

Denver

Jefferson

Using RANGE ( ) in for loop:

numbers = [0, 1, 2, 3, 4]

for num in range(5):

    print(num)

0

1

2

3

4

In this example, we are finding the number of items in the list, then returning each index using range

for i in range(len(counties)):

    print(counties[i])

We can iterate thru a dictionary

Using the For loop we can iterate over a dictionary and get all the keys, all the values, or all the keys and values

counties\_dict = {"Arapahoe": 422829, "Denver": 463353, "Jefferson": 432438}

for county in counties\_dict.keys():

    print(county)

We can also use the keys ( ) method to iterate over a dictionary

keys()

Use the key ‘county’ in the For loop print statement

counties\_dict = {"Arapahoe": 422829, "Denver": 463353, "Jefferson": 432438}

for county in counties\_dict:

    print(counties\_dict[county])

Used .get() method:

counties\_dict = {"Arapahoe": 422829, "Denver": 463353, "Jefferson": 432438}

for county in counties\_dict:

    print(counties\_dict.get(county))

Returns:

422829

463353

432438

Get the Key-Value Pairs of a Dictionary

If we want to print the key-value pair of the dictionary, we use:

items() method in for loop

Syntax:

for key, value in dictionary\_name.items():

print(key, value)

Using the items() method

counties\_dict = {"Arapahoe": 422829, "Denver": 463353, "Jefferson": 432438}

for county, voters in counties\_dict.items():

    print(county, voters)

1st variable declared in the for loop is assigned to the keys, the 2nd is assigned to the values

Iterate through a list of dictionaries – a For loop can be used to iterate through a list of dictionaries like voting\_data list of dictionaries we created earlier. With For loop we can:

1. Retrieve each dictionary in the list
2. Retrieve only the calues of each dictionary
3. Retrieve the key-value pairs of each dictionary

voting\_data = [{"county":"Arapahoe", "registered\_voters": 422829},

                {"county":"Denver", "registered\_voters": 463353},

                {"county":"Jefferson", "registered\_voters": 432438}]

for county\_dict in voting\_data:

    print(county\_dict)

Get Values from a List of Dictionaries:

voting\_data = [{"county":"Arapahoe", "registered\_voters": 422829},

                {"county":"Denver", "registered\_voters": 463353},

                {"county":"Jefferson", "registered\_voters": 432438}]

for county\_dict in voting\_data:

    for value in county\_dict.values():

        print(value)

To retrieve only the values from each dictionary in the list of dictionaries, we need to use a **nested** for loop

First we must use the for loop: for county\_dict in voting\_data:

for county\_dict in voting\_data:

    for value in county\_dict.values():

        print(value)

Printing Formats:

WithOUT F-strings:

my\_votes = int(input("How many votes did you get in the election? "))

total\_votes = int(input("What is the total votes in the election? "))

percentage\_votes = (my\_votes / total\_votes) \* 100

print("I received " + str(percentage\_votes)+"% of the total votes.")

With F-string:

my\_votes = int(input("How many votes did you get in the election? "))

total\_votes = int(input("What is the total votes in the election? "))

print(f"I received {my\_votes / total\_votes \* 100}% of the total votes.")

Dictionaries:

WithOUT Using F-strings:

counties\_dict = {"Arapahoe": 369237, "Denver":413229, "Jefferson": 390222}

for county, voters in counties\_dict.items():

    print(county + " county has " + str(voters) + " registered voters.")

With F-strings:

for county, voters in counties\_dict.items():

    print(f"{county} county has {voters} registered voters.")

Multiline F-strings:

candidate\_votes = int(input("How many votes did the candidate get in the election? "))

total\_votes = int(input("What is the total number of votes in the election? "))

message\_to\_candidate = (

    f"You received {candidate\_votes} number of votes. "

    f"The total number of votes in the election was {total\_votes}. "

    f"You received {candidate\_votes / total\_votes \* 100}% of the total votes.")

print(message\_to\_candidate)

Output:

You received 3345 number of votes. The total number of votes in the election was 23123. You received 14.466115988409808% of the total votes.

Format Float=Point Decimals:

f'{value:{width}.{precision}}'

message\_to\_candidate = (

    f"You received {candidate\_votes:,} number of votes. "

    f"The total number of votes in the election was {total\_votes:,}. "

    f"You received {candidate\_votes / total\_votes \* 100:.2f}% of the total votes.")

Challenge 2:

To facilitate the design process, programmers use pseudocode to create models or flowcharts for their programs

Focus on the overall design of the program

Well defined, logical steps sequentially ordered is an algorithm

Dependencies – are modules and packages, or programming script that someone else has written, that allows you to increase the functional programming of your code, or speed and efficiency

Dependencies can be viewed like Russian nesting dolls:

1. Dependencies are the largest doll, like the Python datetime module.
2. Inside datetime module are functions, classes, or variables, which is the second largest
3. The methods used of functions and classes are in the 3rd

Diagram

Description automatically generated

The first datetime is the datetime module

The second datetime is the datetime class

Then we used the datetime attribute, now(), to get current time

# Import the datetime class from the datetime module

import datetime

# Use the now() attribute on the datetime class to get the present time

now = datetime.datetime.now()

# Print the present time

print("The time right now is ", now)

CSV Module – import csv to the module

import csv

dir(csv)

Using the dir ( ) fundction, we can pass:

1. A Python module, like the CSV module. The dir ( ) function will return all the functions available in the CSV module
2. A variable, like a dictionary {'key':'value'} for the counties\_dict. The dir ( ) function will return all the functions available on that variable
3. A data type, like str ( ). The dir ( ) function will return all the attributes and methods that can be used with the str data type

You may want to import these modules:

1. Random
2. Numpy

# Add the Pandas dependency

import pandas as pd

import os

school\_data\_load = "Resources/schools\_complete.csv"

student\_data\_load = "Resources/students\_complete.csv"

# Connecting the data

school\_data\_load = os.path.join("Resources", "schools\_complete.csv")

student\_data\_load = os.path.join("Resources", "students\_complete.csv")

# Store into a Data Frame

# pd. converts the CSV to dataframe

school\_data\_df = pd.read\_csv(school\_data\_load)

school\_data\_df