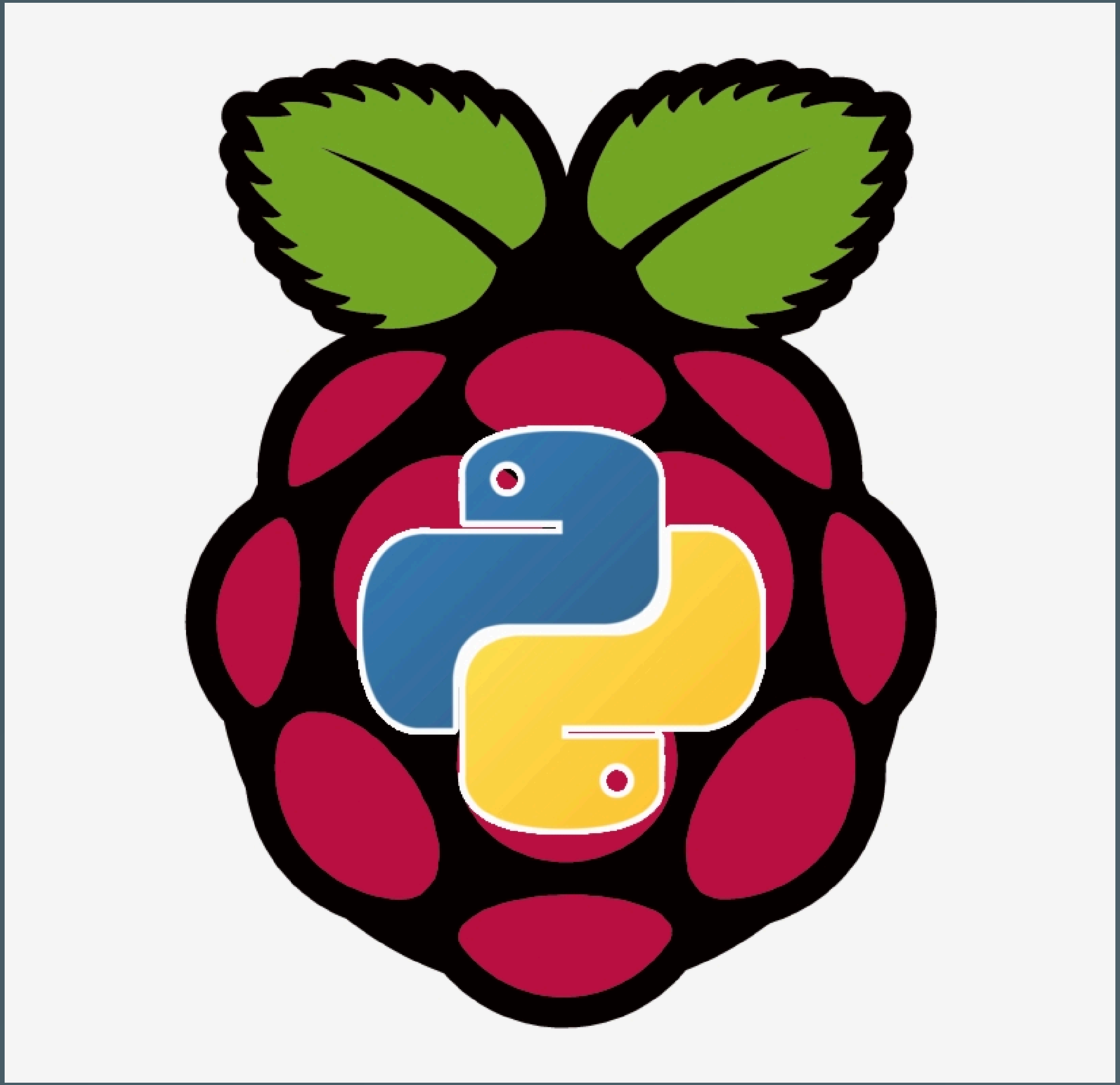
Introduction to Python 🐍

- Lists and Dictionaries
- File IO
- Modules
- Libraries
- And more...

By: Hedron Hackerspace



Lists, Dictionaries, Tuples, and Sets

- `list` - Ordered, mutable, subscriptable object that stores values
- `dict` - Ordered, mutable list of key-value pairs
- `tuple` - Ordered, immutable list of values
- `set` - Unordered, immutable list of values (doesn't allow duplicates)

- Ordered - Values are listed in a specific order
- Mutable/Immutable - Something that can change/cannot change
- Subscriptable - Object is indexed

TLDR Version

- `list` - Group of values
- `dict` - List of key-value pairs
- `tuple` - List, but you can't change anything after its initialized
- `set` - Tuple, but it doesn't allow duplicate values

- There is also a `range`, but its just a list of stepped numbers
  - Ex. `range(10)` is `[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`
  - Ranges will be covered more later (useful in `for` loops)

We will not be extensively covering tuples or sets since they are pretty niche

Lists

```python
# Creating an empty list
empty_list = []

# Creating a list with pre-existing values
random_byte = 0x7F
random_list = ["Python is alright", -286.242, False, random_byte, False]
```

- Can include multiple data types (including more lists)
- Can be as long as you want (just watch out for memory usage)
- Objects can be added, changed, or removed from the list
- Lists can also be sorted, joined (to another list), and emptied

Accessing and Changing Values

```
burrito = ["Flour tortilla", "Cheese", "Black beans", "Rice", "Hot sauce"]
print(burrito[0], burrito[1])
>>> Flour tortilla Cheese

burrito[1] = "Shredded Cheese"
print(burrito[1])
>>> Shredded Cheese
```

- Values in the list can be accessed by specifying a specific index
  - Indices are the position in which the data is located
  - Indices start at 0 and increment by one
- Values can be changed by setting the index equal to something

Negative Indices and Slicing

```
burrito = ["Flour tortilla", "Cheese", "Black beans", "Rice", "Hot sauce"]
# Prints the last value in the list
print(burrito[-1])
>>> Hot sauce

# Prints the first index to the fourth (excluding the last index)
print(burrito[1:4])
>>> ['Cheese', 'Black beans', 'Rice']
```

- Negative indices work in reverse order (and start at 1)
- Slicing refers to selecting a range of indices to use
  - Starting index is inclusive, the ending index is exclusive
  - Empty slice values default to `[beginning:end:step=1]`

Adding and Removing Values

```
# Add "Chicken" to the end of the list
burrito.append("Chicken")
print(burrito)
>>> ['Flour tortilla', 'Cheese', 'Black beans',
     'Rice', 'Hot sauce', 'Chicken']

# Remove the first instance of the value "Chicken"
burrito.remove("Chicken")
print(burrito)
>>> ['Flour tortilla', 'Cheese', 'Black beans', 'Rice', 'Hot sauce']
```

- `append(value: Any)` adds the given value to the end of the list
- `remove(value: Any)` removes the first occurrence of the value

Adding and Removing Indices

```
# Insert "Chicken" at the fourth index
burrito.insert(4, "Chicken")
print(burrito)
>>> ['Flour tortilla', 'Cheese', 'Black beans',
     'Rice', 'Chicken', 'Hot sauce']

# Remove the fourth value from the list
burrito.pop(4)
print(burrito)
>>> ['Flour tortilla', 'Cheese', 'Black beans', 'Rice', 'Hot sauce']
```

- `insert(index: int, value: Any)` adds a value at the indexed position
- `pop(index: int)` removes the value at the given index of a list

Dictionaries

```
dishwasher = {"Forks": 1, "Spoons": 7, "Cutting knife": 4,
              "Butter knife": 3, "Plates": 1, "Bowls": 7}
clean_items = {"Forks": 9, "Spoons": 3, "Cutting knife": 6,
               "Butter knife": 7, "Plates": 14, "Bowls": 8}

print(dishwasher["Plates"])
print(clean_items["Plates"])
>>> 1
>>> 14
```

- Use key-value pairs and are indexed by `str` values
  - Can also use `clean_items.get("Forks")`
- Values can also be anything (including more dictionaries)

Getting Keys and Values

```
print(clean_items.keys())   # Returns a <dict_keys> type
>>> ['Forks', 'Spoons', 'Cutting knife', 'Butter knife', 'Plates', 'Bowls']
print(clean_items.values()) # Returns a <dict_values> type
>>> [9, 3, 6, 7, 14, 8]
print(clean_items.items())  # Returns a <dict_items> type
>>> [["Forks", 9], ["Spoons", 3], ["Cutting knife", 6],
     ["Butter knife", 7], ["Plates", 14], ["Bowls", 8]]
```

- `keys()` returns a list* of keys in the dict
- `values()` returns a list* of values in the dict
- `items()` returns a list* of all key-value pair lists (a list of lists)
- *These will need to be type cast to a `list` to be subscriptable

Adding and Removing Keys

```
# Set "Pans" equal to 4
clean_items.update({"Pans": 4})
print(clean_items["Pans"])
>>> 4

# Remove "Pans" from the dictionary
clean_items.pop("Pans")
print(clean_items["Pans"])
>>> KeyError
```

- `update(entry: dict)` updates the dict entry's value
  - Can use `clean_items["Pans"] = 4`, even if that key doesn't exist
- `pop(entry: str)` deletes an entry in the dict

File IO

- 2 different methods for interacting with files
  1. **Using `open()`:** Built-in Python function for interacting with text-based files (ex. `.txt`, etc.)
  2. **Using another library/module:** Libraries that work with specific file types (ex. `.json`, `.csv`, `.yaml`, etc.)
- Today, we'll be using the first method to keep things simple

File Permission Modes

- 4 different file permission modes
  - `'x'` - Creates the file
  - `'r'` - Reads from the file
  - `'w'` - Overwrites the entire file
  - `'a'` - Appends to the end of the file
- All modes return an error if the file does not exist (inverted with `'x'` mode)

Creating Files

```
file = open("my_recipe.txt", 'x') # Creates a file object
file.close()                    # Closes the file object
```

- `open(file: str, mode: str)` returns a file object
- The file that you would like to edit does not have to be in the same directory as the executing program
- `close()` closes the file object
- The file must be closed when you are done reading from or writing to it to help prevent file corruption

Writing Files

```
file = open("my_recipe.txt", 'w')                  # Open a file called `my_recipe.txt`
file.write("This recipe will be delicious!")  # Write some text to the file
file.close()                                       # Close the file
```

- `write(text: str)` writes the string to the file
- `writelines(text: [str])` writes a list of strings to the file
- **WARNING:** This will **completely overwrite** the file, so be careful and either save it's contents before running or choose `'a'` as the mode

Reading Files

```
file = open("my_recipe.txt", 'r')
recipe = file.readlines(8)
file.close()
print(recipe)
>>> This rec
```

- `read(bytes: int)` reads up to n-bytes of the file, reads the whole file if `bytes` is not specified
- `readline(size: int)` reads up to n-bytes from one line of the file, reads the whole line if `size` is not specified

Libraries/Modules

- Provides additional functionality to your Python program
- Usually installed using `pip`, the main Python package installer
- Library, module, package, extension, and API are often used interchangeably, so don't get confused if you hear any of these
- Python comes with some pre-installed "standard libraries" like `math`, `random`, `os`, `time`, `curses`, `tkinter`, and more
  - You can find a complete list on the Python docs

Installing Libraries

This example installs the Numpy library:

1. Open a terminal window and enter `pip --version`
2. If `pip` is installed properly, type `pip install numpy`
    1. To update the library, add `--upgrade` before listing packages
3. After a minute, you should see a message from `pip` akin to "numpy-<version> has been successfully installed"
4. In the same terminal window, enter the Python interpreter with `python`
5. Test the installation with `import numpy; print(numpy.__version__)`

Using Libraries

```python
import numpy as np    # Import Numpy under the alias `np`

print(np.arange(100)) # Create a range of numbers from 0 to 99
```

- `import numpy` will allow use of the specified library
- `as np` changes the alias that is used from `numpy` to `np`
  - Not very useful in this example, but a great time saver when using libraries extensively
- Numpy is a useful math library that is extremely performant
- This basic program generates and prints a number range from 0-99

Creating Custom Modules

1. First in a new workspace or folder, create two files
    1. `unit_conversion.py` will be your main program file
    2. `units.py` will be your custom made unit conversion module
2. Then choose some units that you want to convert, some ideas include:
    1. Celsius <-> Fahrenheit
    2. Millimeters <-> Inches
    3. Watts <-> Joules
3. Now let's start implementing everything

Creating the Conversion Functions

```
def toFahrenheit(celsius):
    fahrenheit = (celsius * 9/5) + 32
    return fahrenheit

def toCelsius(fahrenheit):
    celsius = (fahrenheit - 32) * 5/9
    return celsius
```

- In the `units.py` file, we are going to create a conversion function for both Fahrenheit and Celsius
- Each function simply returns the converted value
- Variables, functions, and objects can also be referenced from here

Using Your Module

```python
import units

temp_f = 68.0
temp_c = 15.0

conv_f = units.toCelsius(temp_f)
conv_c = units.toFahrenheit(temp_c)

print(f"Celsius: {temp_c}C/Converted: {conv_c}F")
print(f"Fahrenheit: {temp_f}F/Converted: {conv_f}C")
```

- `import units` will run all of the Python code from your module
- All variables, functions, and objects are references with `units`