CIS 112

Intro to Programming Using Python

Module 3 Part 2

Agenda for the Day!

- Modules
- Serialization
- Exceptions and Error Handling

Modules

So what is a module?

- Back to our theme of modularity and scaling!
  - We've explored multiple ways to segment your code and create more modular and flexible programs, but at some point even that isn't enough. You'll have too many operations and methods nested within your main program to easily account for.
  - So as a next level of modularity, we need to explore segmenting your program across different files.
- To support this, Python has a way to put function definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a **module**; definitions from a module can be **imported into other modules or into the main module** (the collection of variables that you have access to in a script executed at the top level and in calculator mode).
  - Note we've been using code libraries almost from the start, we never really bothered to create our own, until now!
- The file name is the module name with the suffix .py appended. Within a module, the module's name (as a string) is available as the value of the global variable __name__.

So how do we create and use?

- Create function definitions within a script file and save as a .py extension
- Then in another file, add an import statement referencing the file name
- Module methods can then be invoked using the ‘module.method()’ terminology
  - Note! The module files must be co-located with the current script to be imported properly
- We can also use aliases to import

Save this code in a file named `mymodule.py`

```
def greeting(name):
  print("Hello, " + name)
```

```
import mymodule

mymodule.greeting("Jonathan")
```

```
import mymodule as mx

a = mx.person1["age"]
print(a)
```

More Modules

- Modules can also store attributes as well that can be similarly imported
- It should also be noted that we can opt to import only part of a module by invoking the **from** keyword

Save this code in the file `mymodule.py`

```
person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

```
import mymodule

a = mymodule.person1["age"]
print(a)
```

```
from mymodule import person1

print (person1["age"])
```

Serialization

Serialize me!

- What is data serialization?
  - Data serialization is the process of converting **structured data** to a format that allows **sharing** or **storage** of the data in a form that allows **recovery** of its original structure.
  - In some cases, the secondary intention of data serialization is to **minimize the data's size** which then reduces disk space or bandwidth requirements.
- **Flat** vs. **Nested data**
  - Before beginning to serialize data, it is important to identify or decide how the data should be structured during data serialization - flat or nested.
  - The differences in the two styles are shown in the below examples:
    - Flat:

```
{ "Type" : "A", "field1": "value1", "field2": "value2", "field3": "val
```

    - Nested:

```
{"A"
    { "field1": "value1", "field2": "value2", "field3": "value3" } }
```

Flat Data methods

- **CSV File**
  - The CSV module in Python implements classes to read and write tabular data in CSV format.

```python
# Reading CSV content from a file
import csv
with open('/tmp/file.csv', newline='') as f:
    reader = csv.reader(f)
    for row in reader:
        print(row)
```

```python
# Writing CSV content to a file
import csv
with open('/temp/file.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerows(iterable)
```

YAML

- YAML is a recursive acronym that stands for **YAML Ain’t Markup Language.**
- It is designed with flexibility and accessibility in mind, so it works with all modern programming languages and is widely used for cross-data sharing.
- All of these factors contribute to YAML’s popularity as a configuration language in the DevOps domain, where it is widely used with well-known tools such as Kubernetes, Ansible, Terraform, and many others.
- While not a natively supported datastructure, a number of 3rd party libraries support reading in or out YAMLs

```python
# Reading YAML content from a file using the load method
import yaml
with open('/tmp/file.yaml', 'r', newline='') as f:
    try:
        print(yaml.load(f))
    except yaml.YAMLError as ymlexcp:
        print(ymlexcp)
```

```yaml
---
# A sample yaml file
company: spacelift
domain:
 - devops
 - devsecops
tutorial:
  - yaml:
      name: "YAML Ain't Markup Language"
      type: awesome
      born: 2001
  - json:
      name: JavaScript Object Notation
      type: great
      born: 2001
  - xml:
      name: Extensible Markup Language
      type: good
      born: 1996
author: omkarbirade
published: true
```

JSON

- What is JSON?
  a. JSON stands for JavaScript Object Notation
  b. It is a lightweight data-interchange format
  c. It is plain text written in JavaScript object notation
  d. It is used to send data between computers
  e. JSON is language independent
- Python's JSON module can be used to read and write JSON files. Example code below:


```
{
  "squadName": "Super hero squad",
  "homeTown": "Metro City",
  "formed": 2016,
  "secretBase": "Super tower",
  "active": true,
  "members": [
    {
      "name": "Molecule Man",
      "age": 29,
      "secretIdentity": "Dan Jukes",
      "powers": ["Radiation resistance", "Turning tiny", "Radiation blast"]
    },
    {
      "name": "Madame Uppercut",
      "age": 39,
      "secretIdentity": "Jane Wilson",
      "powers": [
        "Million tonne punch",
        "Damage resistance",
        "Superhuman reflexes"
      ]
    },
```


```python
# Reading JSON content from a file
import json
with open('/tmp/file.json', 'r') as f:
    data = json.load(f)
```

```python
# Writing JSON content to a file using the dump method
import json
with open('/tmp/file.json', 'w') as f:
    json.dump(data, f, sort_keys=True)
```

Pickle!

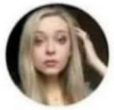
- The native data serialization module for Python is called Pickle.
  - The pickle module implements binary protocols for serializing and de-serializing a Python object structure.
  - “Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream is converted back into an object hierarchy.
- **JSON vs. Pickle**
  - JSON is a text serialization format (it outputs unicode text, although most of the time it is then encoded to utf-8), while pickle is a binary serialization format
    - JSON is human-readable, while pickle is not;
  - JSON is interoperable and widely used outside of the Python ecosystem, while pickle is Python-specific;
  - Unlike pickle, deserializing untrusted JSON does not in itself create an arbitrary code execution vulnerability.

```
import pickle

#Here's an example dict
grades = { 'Alice': 89, 'Bob': 72, 'Charles': 87 }

#Use dumps to convert the object to a serialized string
serial_grades = pickle.dumps( grades )

#Use loads to de-serialize an object
received_grades = pickle.loads( serial_grades )
```

Error Handling

To err is human...

- You try everything you can, but inevitably the user will do the unexpected.
- Rather than crashing your program, you want to have the ability to catch errors and report them
- Wrapping a code block in a try/except block allows you to do precisely that!



anja
@internetanja

love when the washing machine gets to the angry part. let it out girl

```
try:
    number = int(input())
    print(number **2)
except:
  print('Unable to square non-numeric input')
```

```
two
Unable to square non-numeric input
```

We can get picky on our errors too...

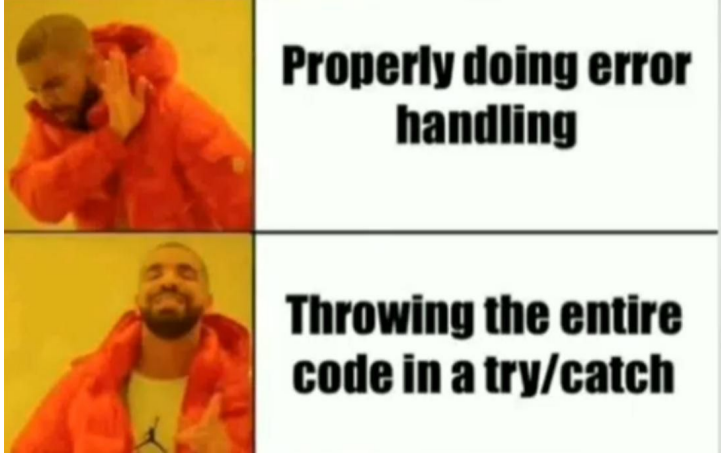

```
try:
  number = int(input())
  print(2/number)
except ValueError:
  print('Unable to square non-numeric input')
except ZeroDivisionError:
  print('Unable to divide a number by 0')
```

```
0
Unable to divide a number by 0
```

- You can also specify your output by declaring specific exception error types and customizing the response to the user!

Or repurpose existing error-types (or make some of your own)!

- Sometimes your program won't technically crash, but still won't produce an optimal result
- You can use Python's built in error-handling to raise custom errors resulting from program outputs you wish to avoid
- You can also create a custom exception class to raise




```
user_input = ''
while user_input != 'q':
    try:
        weight = int(input('Enter weight (in pounds): '))
        if weight < 0:
            raise ValueError('Invalid weight.')

        height = int(input('Enter height (in inches): '))
        if height <= 0:
            raise ValueError('Invalid height.')

        bmi = (float(weight) * 703) / (float(height * height))
        print(f'BMI: {bmi}')
        print('(CDC: 18.6-24.9 normal)\n')
        # Source www.cdc.gov

    except ValueError as excpt:
        print(excpt)
        print('Could not calculate health info.\n')

    user_input = input("Enter any key ('q' to quit): ")
```

Functions and Errors

```
def get_weight():
    weight = int(input('Enter weight (in pounds): '))
    if weight < 0:
        raise ValueError('Invalid weight.')
    return weight

def get_height():
    height = int(input('Enter height (in inches): '))
    if height <= 0:
        raise ValueError('Invalid height.')
    return height

user_input = ''
while user_input != 'q':
    try:
        weight = get_weight()
        height = get_height()

        bmi = (float(weight) / float(height * height)) * 703
        print(f'BMI: {bmi}')
        print('(CDC: 18.6-24.9 normal)\n')
        # Source www.cdc.gov

    except ValueError as excpt:
        print(excpt)
        print('Could not calculate health info.\n')

    user_input = input("Enter any key ('q' to quit): ")
```

```
Enter weight (in pounds): 150
Enter height (in inches): 66
BMI: 24.207988980716255
(CDC: 18.6-24.9 normal)

Enter any key ('q' to quit): a
Enter weight (in pounds): -1
Invalid weight.
Could not calculate health info.

Enter any key ('q' to quit): a
Enter weight (in pounds): 150
Enter height (in inches): -1
Invalid height.
Could not calculate health info.

Enter any key ('q' to quit): q
```

Janet Forklift
@janetforklift

The moment I became an adult was when I looked at a bag and thought, “Wow, this bag would be a great bag to put other bags in.”

- Embedding error handling within functions can allow you to pass through more detailed error tracking information to the overall program if it’s handling a try except statement!

And Finally...

say it ain't so,
i will not go
turn the lights off,
carry me home

- So say your code trips an exception, good news is your code doesn’t necessarily crash, but other than throwing an exception for the block, is there anything else we can do?
- Finally code allows us to run a code block regardless of whether or not an exception is raised.