ECE 449/590 – OOP and Machine Learning
Lecture 02 Python

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

August 24, 2022

Reading Assignment

- This lecture: Python Introduction
- Next lecture: Project Introduction

Python

- Python 3
  - Python was created in '90s
  - Python 2, a very popular version, was released in 2000 and was officially discontinued in 2020, long after Python 3 was released in 2008
- Multiple methods to run Python code.
  - Use the Python Interpreter from command line.
  - Use Jupyter notebook from a Web browser.
    - Popular among data scientists and AI researchers.
    - E.g. via Colab `https://colab.research.google.com/`
  - Via `.py` programs.
    - Popular among software developers.
    - Our projects will use this method.

Hello World

```
>python3
Python 3.x.x
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello world!")
Hello world!
>>> exit()
```

- Start the Python Interpreter from command line using `python3`.
- Use function `print` to print strings.
- Use function `exit` to exit the interpreter.
- `;` can be used but are usually not used.

Variable

```
>>> a = 1+2*3
>>> print(a)
7
>>> a = "hello"
>>> print(a)
hello
```

- Dynamic typing: a variables has a type but the type can be changed by assigning a value of different type.
- `print` can work with all types of Python variables.

List

```
>>> b = [1, "a", 2, "b", 3, "c"]
>>> print(len(b), b)
6 [1, 'a', 2, 'b', 3, 'c']
>>> print(b[0])
1
```

- Use `[]` to define a Python list.
  - More like Java's `ArrayList` or C++'s `std::vector` than linked list.
  - Elements can have different types.
- Use function `len` to get size/length of lists.
- Use `[]` to access elements in lists using 0-based indices.

Dictionary

```
>>> c = {"a": 1, 2: "b", "c": 3}
>>> print(len(c), c)
3 {'c': 3, 2: 'b', 'a': 1}
>>> print(c["a"])
1
```

- Use `{}` to define a Python dictionary.
  - Key-value pairs with unique keys.
  - Keys are not sorted – similar to Java's `HashMap` or C++'s `std::unordered_map`.
- Use function `len` to get size and use `[]` to retrieve value via key.

Tuple

```
>>> d = (1, "b", 3)
>>> print(len(d), d)
3 (1, 'b', 3)
>>> print(d[1])
b
```

- Use `()` to define a Python tuple.
  - A "frozen" list that cannot be modified.

Slicing

```
>>> a = [1,2,3,4]
>>> print(a[1:3], a[:-1])
[2, 3] [1, 2, 3]
>>> s = "Hello world!"
>>> print(s[:5], s[-6:])
Hello world!
```

- Use `[begin:end]` to obtain a slice of list, string, or tuple.
  - Half close half open (`begin` included, `end` excluded).
  - `begin = 0` if omitted, `end = len()` if omitted.
- Negative indices count from the back.
  - E.g. `-1` refers to the last element.
  - Can also be used to refer to an element instead of a slice.

String Formatting

```
>>> a = 1.10001
>>> b = 10
>>> c = "hello"
>>> print("%.3f-%05d-%s" % (a, b, c))
1.100-00010-hello
>>> d = [1, 2, 3]
>>> print("List d is %s." % d)
List d is [1, 2, 3].
```

- printf format string is supported via `%`.
- Use a tuple if there are more than one arguments.
- Use `%s` to convert any Python variable into a string.

Getting Help

```
>>> a = []
>>> dir(a)
['__add__', '__class__', ..., 'pop', 'remove', 'reverse', 'sort']
>>> help(a.sort)
Help on built-in function sort:

sort(...) method of builtins.list instance
    L.sort(key=None, reverse=False) -> None -- stable sort *IN PLACE*
```

- Use function `dir` to list all methods from a Python variable.
- Use function `help` to display a short description of a method.

Working with .py Programs

```
# date.py
import datetime

print(datetime.date.today())
```

- Execute the program with `python3 date.py`.
- The program runs line by line.
  - There is no `main` function.
- Use `import` to include libraries for additional functionalities.

Loops

```
a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
for j in a:
  print(j)
for i in range(10):
  print(i)
```

- The `for in` loops are used most frequently.
  - Don't forget the `:` at the end of line.
- Use function `range` if you just need a list of numbers.
- Proper indentation is mandatory.
  - There is no limitation on size of indentations, like 2 spaces, 4 spaces, or 1 tab, though they need to be consistent throughout the whole file.
- `while` loops are also supported.

Branches

```
import random

d = random.random()
if d < 0.4:
  print("Win!")
elif d > 0.6:
  print("Lose!")
else:
  print("Tie!")
```

- Don't forget the `:` at the end of `if`, `elif`, and `else`.
- Use `and`, `or`, `not` to combine conditions.
- Use indentations to highlight the block for each branch.

Functions

```
def add(a, b):
  return a+b

print(add(1, 1))
```

- Provide a function name and a list of parameters to define a function.
- Use indentations to highlight the block for the function body.
- Note that we don't specify the types of the parameters.

Classes

```
class Map:
  def __init__(self):
    self.data = {}
  def put(self, k, v):
    self.data[k] = v
  def get(self, k):
    return self.data[k] if k in self.data else None

a = Map()
a.put("x", 1)
print(a.get("x"), a.get("y"))
```

- Data members are created in the constructor `__init__`.
- The first parameter of the member functions is usually named `self` to refer to the object itself.
  - Similar to `this` but use of `self` is mandatory.
- Use the class name to create an object.
  - There is no `new`.

List Comprehension

```
squares = [i*i for i in range(10)]
odd_squares = [i*i for i in range(10) if i%2 == 1]
even_odd = [(i, j)
            for i in range(10) if i%2 == 0
            for j in range(10) if j%2 == 1]
```

- `[new_item for item in list if cond]`
  - Create a list from another without writing a loop.
  - `if` clause may be omitted.
- Multiple `for ... in ...`'s may be used to create a list from the Cartesian product of multiple lists

Polymorphism

```
def add(a, b):
  return a+b

print(add(1, 1))
print(add("Hello ", "world!"))
print(add([1, 2], [3, 4]))
```

- Same code works differently for different types.

Polymorphism via Duck Typing

```
class A:
  def show(self):
    print("This is type A.")

class B:
  def show(self):
    print("This is type B.")

def display(x):
  x.show()

display(A())
display(B())
```

- "If it walks like a duck and it quacks like a duck, then it must be a duck."
- There is no need to define a base class or an interface to make use of polymorphism in Python.
- The function `display` works as long as the parameter `x` has a member function with the name `show`

Summary

- More Python and NumPy tutorials.
  - `https://docs.python.org/3/tutorial/index.html`
  - `https://cs231n.github.io/python-numpy-tutorial/`