Deep Learning Toolkit *(Python)*

Rowel Atienza, PhD
University of the Philippines
github.com/roatienza
2023

Python

https://github.com/dabeaz-course/practical-python

Python

Scripting interpreted language

Exercise: activate python on your terminal

Exercise: create a new python source file in `vscode`

Numbers

No need to declare the data type but common data types are supported : Boolean to complex numbers

Exercise:

- Generate 10 random integers. Store in a list. Print.
- Print the min and max
- Print in ascending order

Supports data type cast like in C

Exercise:

- Generate 10 random floats. Store in a list. Print.
- Convert all floats to int. Print.

All of these exercises can now be solved by Copilot Chat/. See the demo!

Strings

Declared using single or double quotes

```
name = "deep learning is fun"
```

Can be indexed

```
print(name[5:])
```

Can be concatenated

```
print(name + "!")
```

Supports string manipulation

```
print(name.replace("deep", "machine"))
```

Search

```
print("learn" in name)
```

String functions

```
print(name.upper())
```

None

None is used as a placeholder for unsure or missing data type or value

```
email_address = None
```

List

A **list** is a data structure that is a mutable, or changeable, ordered sequence of elements

Zero or more elements that are separated by commas

```
x = [1, "fox", 3.4, [8, 16]]
```

Indexed

```
print(x[1])
```

Concatenate

```
y = [1, 2, 3, 4, 5]
z = [1, 4, 9, 16, 25, 36]
y + z
```

Append

```
y.append(6)
```

List - Slicing

```
y[start:end:interval]
```

```
y[0:4:2]
y[::3]
y[::-1]
```

Loops

for

```
>>> x = [1, "fox", 3.4, [8, 16]]
>>>
>>> for i in x:
...     print(i)
...
1
fox
3.4
[8, 16]
```

while

```
>>> i = 0
>>> while i < len(x):
...     print(x[i])
...     i += 1
...
1
fox
3.4
[8, 16]
```

Function

We use the `def` keyword to define a function

A function has 0 or more inputs. Same with outputs.

Example: given a list of integers, get all even integers, store in a new list and print.

```
y = [8, 1, 4, 2, 0, 7, 5, 6, 3]
def filter_even(x):
    result = []
    for i in x:
        if i % 2 == 0:
            result.append(i)
    return result

print(filter_even(y))
```

Object Oriented

Class and inheritance

Methods and properties

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return f"{self.name} is {self.age} years old."


x = Person("John", 30)
print(x)
```

Object Oriented - PyTorch

Our deep learning models will be built using OO techniques

```
import torch

class GNet(torch.nn.Module):
    def __init__(self, mean=0., std=1.):
        super(GNet, self).__init__()
        self.mean = torch.Tensor([mean])
        self.std = torch.Tensor([std])

    def forward(self, x):
        return x*torch.normal(mean=self.mean, std=self.std)

x = GNet()
print(x(3))
```

Type hints

- Not required but helps model compilers figure out data types

```
def greeting(name: str) -> str:
    return 'Hello ' + name
```

Decorators

- The @ decorator is a design pattern that allows you to modify the functionality of a function by wrapping it in another function.

```python
def log_decorator(func):
    def wrapper(*args, **kwargs):
        print(f"Calling function {func.__name__} with args {args} and kwargs {kwargs}")
        result = func(*args, **kwargs)
        print(f"Function {func.__name__} returned {result}")
        return result
    return wrapper

@log_decorator
def add_numbers(a, b):
    return a + b

result = add_numbers(2, 3)
print(result)
```

Out:

```
Calling function add_numbers with args (2, 3) and kwargs {}
Function add_numbers returned 5
5
```

Reference

Practical python https://github.com/dabeaz-course/practical-python

End