CS 357 - 02 Python

Boyang Li (boyangl3)

Data types:

```
a = 2           # a is int
b = 3.0         # b is float
c = a + b       # c is float
d = 2 * a       # d is int
```

Names, values and modifying an object:

```
a = [1, 2, 3]       # define a as a Python list
b = a               # b = [1, 2, 3]
b.append(4)         # b = [1, 2, 3, 4]
print(a)            # [1, 2, 3, 4]
```

Memory management:

```
a = [1, 2, 3]
b = [1, 2, 3]
print("IS", a is b)        # "is" is used to compare ids; False
print("EQUAL", a == b)     # "==" is used to compare values; True

c = [1, 2, 3]
d = a                      # Both "is" and "==" is True
```

Mutable and immutable objects:

```
# Mutable objects: can be modified after created
# Example: Python lists
myList = [3, 5, 7]
myList[0] = 1          # myList = [1, 5, 7]

# Tuples and strings are immuatble objects
# the operations would result in a type error
myTuple = (3, 5, 7)
myTuple[0] = 1
myString = "357"
myString[0] = '1'
```

List operaitons:

```
a = [1, 2, 3]
b = a
a is b                  # True

a = a + [4]             # a = [1, 2, 3, 4] and b = [1, 2, 3]
a is b                  # False

# Instead, take following operation:
a += [4]                # a = b = [1, 2, 3, 4]
a is b                  # True
```

Advanced naming:

```
fruit = 'apple'

lunch = []
lunch.append(fruit)       # lunch = ['apple']

dinner = lunch
# dinner = ['apple'], lunch = ['apple']

dinner.append('fish')
# dinner = ['apple', 'fish'], lunch = ['apple', 'fish']

fruit = 'pear'

meals = [fruit, lunch, dinner]
# meals = ['pear', ['apple', 'fish'], ['apple', 'fish']]
```

Indexing:

```
# Given a list a, the formatting indexing: a[i:j:k]
# i - starting index; j - ending index (exclusive); k - step size

a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
a[1::2][::-1]
# output = [9, 7, 5, 3, 1]
# First bracket starting from index 1 to the end, step size 2
# Second bracket with step size -1, reverse the index
```

Control flow:

```
# Traditional way
mylist = []
for i in range(50):
  if i % 7 == 0:
    mylist.append(i**2)

# mylist = [0, 49, 196, 441, 784, 1225, 1764, 2401]

# Enhanced way
mylist2 = [i**2 for i in range(50) if i % 7 == 0]

# mylist2 has the same value as mylist
```

Function scope:

```
def add_minor(person):
    person.append('math')

def switch_majors(person):
    person = ['physics']
    person.append('economics')

    # here, the person variable has a local scope,
    # meaning its value/update only happens within the function,
    # and is "ignored" outside of it

# all variables created out here have a global scope
John = ['computer_science']
Tim = John
add_minor(Tim)
switch_majors(John)
print(John, Tim)
# ['computer_science', 'math'] ['computer_science', 'math']
```

Dictionary:

```python
# definition of a dictionary
myDict = {
    "number": 357,
    "major": "computer science",
    "credit hours": 3
}

# dictionaries are mutable, thus can be modified,
# either modifying an existing value or creating a new key
myDict["major"] = "math"
myDict["level"] = "undergrad"

# looping through the keys or the values over the dictionary
for x in myDict:
  print(x)
  print(myDict[x])

for x in myDict.values():
  print(x)

# there are multiple ways to remove an existing entry
myDict.pop("level")

# copy an existing dictionary into a new reference
anotherDict = myDict.copy()
```

Type annotations:

```python
# simple variable declaration (the type we're used to)
a = 5

# type annotated variable declaration
a: int = 5

# simple function definition (the type we're used to)
def add_two_numbers(a, b):
  return a + b

# type annotated function definition (both inputs and output)
def add_two_numbers(a: int, b: int) -> int:
  return a + b
```

Numpy array initialization and operation:

```
import numpy as np
# 2d array of zeros, shape 2 x 2
np.zeros((2, 2))

# 2d array of ones, shape 2 x 2
np.ones((2, 2))

# array of numbers evenly spaced between 2 and 3 (4 entries)
np.linspace(2, 3, 4)
#array([2, 2.333, 2.667, 3])

# 2d array of random numbers between 0 and 1, shape 2 x 2
np.random.rand(2, 2)

# empty 2 x 2 2d array
np.empty((2, 2))

a = np.zeros((2, 2))
print(a.shape)        # will return (2, 2)
print(a.dtype)        # returns float
a = a.astype(int)
print(a.dtype)        # returns int
b = a.copy()          # b is deep copy of a
```

Numpy array indexing and slicing:

```
import numpy as np
a = np.array([3, 7, 9, 10, 3, 5])
b = np.array([[1, 2, 3], [4, 5, 6]])

print(a[2])     # prints 9
print(b[0, 0])  # prints 1

# slicing examples for both 1d and 2d NumPy arrays
# (for 2d arrays we specify both the row and col)
print(a[1:3])     # prints [7, 9]
print(b[0:1, 2])  # prints [3]

# If we leave the row/col index empty or use a colon(:)
# then we're saying that we select the entire row/col
# this assumes the starting index of the row is 0,
# so we select the entire first row, prints [[1, 2, 3]]
print(b[:1])
```

Array manipulation:

```
import numpy as np
a = np.array([3, 7, 9, 10, 3, 5])
b = np.array([[1, 2, 3], [4, 5, 6]])

b = np.reshape(b, (6, 1))        # Make b as 1d array
a = np.reshape(a, (3, 2))        # Make a as 2d array

a = a.flatten()                  # Flatten a directly

a_transpose = a.T                # Transpose
```

Array mathematics and linear algebra:

```
import numpy as np
import numpy.linalg as la

a = np.array([[8, 9]])
b = np.array([[1, 2, 3], [4, 5, 6]])

# The most important operation you'll do is matrix multiplication
# we can do this easily in 2 ways (both are the same)
c = np.dot(a, b)
c = a @ b

# We can do a lot of other operations
d = np.sin(a)       # applies to every entry in a
e = np.cos(a)       # applies to every entry in a
f = np.exp(a)       # applies to every entry in a

g = np.sum(a)       # g is a float
h = np.mean(a)      # h is a float
i = np.min(a)       # i is a float

A = np.array([1, 2], [3, 4])
b = np.array([5, 6])

matrix_inv = la.inv(A)           # inverse matrix

eigval, eigvec = la.eig(A)       # eigenvalues and eigenvectors

vec_norm = la.norm(A)            # calculate norm of A
vec_norm_3 = la.norm(A, 3)       # calculate 3-norm of A

x = la.solve(A, b)               # solve lienar system Ax = b
```

Random numbers:

```
import numpy as np

a = np.random.rand(3, 2)
# 3 x 2 array of random numbers 0 to 1

b = np.random.randint(100)
# random int from 0 to 100
x = np.random.randint(100, size=(5))
# array of size 5 with random number 0 to 100
x = np.random.uniform(a, b)
# random number uniformly distributed between a and b


c = np.random.choice([1, 2, 3, 4])
# will randomly return one of the values within the array
```

Broadcasting:

```
import numpy as np

A = np.array([[1, 2, 3, 4, 5]])
B = np.array([
    [10, 20, 30, 40, 50],
    [60, 70, 80, 90, 100],
    [110, 120, 130, 140, 150],
    [160, 170, 180, 190, 200]
])

C = B + A

print(C)
# C = np.array([[ 11,  22,  33,  44,  55],
#         [ 61,  72,  83,  94, 105],
#         [111, 122, 133, 144, 155],
#         [161, 172, 183, 194, 205]])
```

Packages will be useful in CS 357:

1. **NumPy:** https://numpy.org/doc/stable/reference
2. **SciPy:** https://docs.scipy.org/doc/scipy/reference/
3. **SymPy:** https://docs.sympy.org/latest/reference
4. **matplotlib:** https://matplotlib.org/3.5.3/api
5. **Pandas:** https://pandas.pydata.org/docs/reference