Python Fundamentals

DS 8015

Outline

Why Python?

“Hello World” in Java

```
public class HelloWorld {
    public static void main(String[] args){
        System.out.println("Hello World!");
    }
}
```

“Hello World” in C++

```
#include <iostream>
using namespace std;

int main(){
   cout << "Hello World!" << endl;
}
```

“Hello World” in Python

```
print('Hello World!')
```

Who uses Python?

Python at Ryerson:

- Graduate courses...

Python in Business

Python in Business
Dropbox
yelp
Eventbrite
venmo
Quora
Google
Instagram
YouTube

Python Basics

HOW TO INSTALL PYTHON?

- Install Anaconda:

  https://www.anaconda.com/distribution/

- Online python editors:
  - Google Colab
  - Project Jupyter
  - ...
- Manual installation of python:

  https://www.python.org/downloads/

- Other IDEs:
  - Atom
  - Visual Studio Code
  - ...

How to get new tools/libraries?

- pip is the preferred Python package manager. Use pip!

  ```
  pip install numpy
  ```

- When you can, use pip instead of:
  - conda - less flexible, less supported by the community
  - easy_install - the old way to install packages
  - python setup.py install - build package from source

Basic subjects

(1) Interactive Interpreter

(2) Comments

(3) Variables and Types

(4) Numbers and Booleans

(5) Strings and Lists

(6) Console I/O

(7) Control Flow

(8) Loops

(9) Functions

(1) Interactive Interpreter

```
C:\Users>python
Python 3.10.0 (v3.10.0:b494f5935c, Oct  4 2021, 14:59:20)..
Type "help", "copyright", "credits" or "license" for...
>>>
```

⇒ You can write Python code after >>>

- Immediate gratification
- Sandboxed environment to experiment with Python
- Shortens code-test-debug cycle to seconds
- Interactive interpreter is your best friend!

(2) Comments

```
# Single line comments start with a '#'

"""
Multiline comments can be written between
three "s and are often used as function
and module comments.
"""
```

(3) Variables

```
x = 2 # semicolon not needed!
x*7 # => 14

x = "Hello, I'm"
x+"Python!" # => 'Hello, I'm Python'
```

- Where is my type?

  ⇒ int x = 0;

  - Variables in Python are dynamically-typed: declared without an explicit type
  - However, objects have a type, so Python knows the type of a variable, even if you don't

VARIABLE TYPES

```
type(1) # => <class 'int'>
type("Hello") # => <class 'str'>

type(None) # => <class 'NoneType'>
type(int) # => <class 'type'>
type(type(int))# => <class 'type'>
```

(4) Numbers and Math

- Python has two numeric types: int and float

```
3 # => 3 (int)
3.0 # => 3.0 (float)

1 + 1 # => 2
8 - 1 # => 7
10 * 2 # => 20
5 / 2 # => 2.5
13 / 4 # => 3.25
9 / 3 # => 3.0
7 / 1.4 # => 5.0

7 // 3 # => 2 (integer division)
7 % 3 # => 1 (integer modulus)
2 ** 4 # => 16 (exponentiation)
```

BOOLEANS

- bool is a subtype of int, where True == 1 and False == 0

```
True # => True
False # => False

not True # => False
True and False # => False
True or False # => True (short-circuits)

1 == 1 # => True
2 * 3 == 5 # => False
1 != 1 # => False
2 * 3 != 5 # => True

1 < 10 # => True
2 >= 0 # => True
1 < 2 < 3 # => True (1 < 2 and 2 < 3)
1 < 2 >= 3 # => False (1 < 2 and 2 >= 3)
```

(5) Strings

```
# No char in Python! Both ' and " create string literals.

greeting = 'Hello'
group = "world" # Unicode by default
greeting + ' ' + group + "!" # => 'Hello world!'

#INDEXING
s = 'Arthur'
s[0] == 'A'
s[1] == 'r'
s[4] == 'u'
s[6] # Bad! (RaiseError)

#NEGATIVE INDEXING
s[-1] == 'r'
s[-2] == 'u'
s[-4] == 't'
s[-6] == 'A'
```

SLICING

```
s = 'Arthur'

s[0:2] == 'Ar'
s[3:6] == 'hur'
s[1:4] == 'rth'

# implicit start/end
s[:2] == 'Ar'
s[3:] == 'hur'

# passing a step size / reversing strings
s[1:5:2] == 'rh'
s[4::-2] == 'utA'
s[::-1] == 'ruhtrA'
```

Converting values

```
str(42) # => "42"

int("42") # => 42

float("2.5") # => 2.5

float("1") # => 1.0
```

- All objects have a string representation
- Especially useful for reading from file!

LISTS

```
easy_as = [1,2,3]
```

- Square brackets delimits lists
- Commas separate elements
- Equivalent to ArrayList/vector

```
# Create a new list
empty = []
letters = ['a', 'b', 'c', 'd']
numbers = [2, 3, 5]
# Lists can contain elements of different types
mixed = [4, 5, "seconds"]
# Append elements to the end of a list
numbers.append(7) # numbers == [2, 3, 5, 7]
numbers.append(11) # numbers == [2, 3, 5, 7, 11]
```

INSPECTING LIST ELEMENTS

```
letters = ['a', 'b', 'c', 'd']
numbers = [2, 3, 5, 7, 11]

# Access elements at a particular index
numbers[0] # => 2
numbers[-1] # => 11

# You can also slice lists - the same rules apply
letters[:3] # => ['a', 'b', 'c']
numbers[1:-1] # => [3, 5, 7]
```

NESTED LISTS

```
letters = ['a', 'b', 'c', 'd']
numbers = [2, 3, 5, 7, 11]

# Lists really can contain anything
# even other lists!
combo = [letters, numbers]
combo # => [['a', 'b', 'c', 'd'], [2, 3, 5, 7, 11]]
combo[0] # => ['a', 'b', 'c', 'd']
combo[0][1] # => 'b'
combo[1][2:] # => [5, 7, 11]
```

General Queries

```
# Length (len)
len([]) # => 0
len("python") # => 6
len([4, 5, "seconds"]) # => 3

# Membership (in)
0 in [] # => False
'y' in "python" # => True
"minutes" in [4, 5, "seconds"] # => False
```

(6) Console I/O

```
# Read a string from the user
# input() prompts the user for input
>>> name = input("What is your name? ")
# What is your name?

>>> print("I'm Python. Nice to meet you,", name)
# I'm Python. Nice to meet you, Sam

# print() can be used in many different ways
```

(7) Control Flow

```
# parantheses not needed
# no curly braces,but colon (:)
if the_world_is_flat:
    print("Don't fall off!")

# use 4-spaces for indentation (can be customized)
```

Zen of Python: Readability counts

ELIF AND ELSE

```python
if some_condition:
   print("Some condition holds")
elif other_condition:
   print("Other condition holds")
else:
   print("Neither condition holds")

# else is optional
# Python has no switch statement opting for
#  if/elif/else chains
```

PALINDROME?

```
# Palindromes are Spelled the same backwards/forwards
# Is a user-submitted word a palindrome?

word = input("Please enter a word: ")
reversed_word = word[::-1]
if word == reversed_word:
   print("Hooray! You entered a palindrome")
else:
   print("You did not enter a palindrome")
```

Truthy and Falsy

```
# 'Falsy' values
bool(None) # => False
bool(False) # => False
bool(0) # => False
bool(0.0) # => False
bool('') # => False

# Empty data structures are 'falsy'
bool([]) # => False

# Everything else is 'truthy'
bool(41) # => True
bool('abc') # => True
bool([1, 'a', []]) # => True

bool([False]) # => True
bool(int) # => True
```

Checking for Truthiness

```
# How should we check for an empty list?
data = []
...
if data:
    process(data)
else:
    print("There's no data!")

# You should almost never test if expr == True
```

(8) Loops

```
# Loop explicitly over data
for item in iterable:
   process(item)

# iterable can be Strings, lists, etc.
# No loop counter
```

Looping over Strings and Lists

```python
# Loop over characters in a string.
for ch in "DS8015":
    print(ch)
# Prints D, S, 8, 0, 1, and 5

# Loop over elements of a list.
for number in [3, 1, 4, 1, 5]:
    print(number ** 2, end='|')
# => 9|1|16|1|25|
```

Compare it with Java:

```java
String s = "DS8015";
for (int i = 0; i < s.length(); ++i) {
    char ch = s.charAt(i);
    System.out.println(ch);
}
```

RANGE

```
range(3)
# generates 0, 1, 2

range(5, 10)
# generates 5, 6, 7, 8, 9

range(2, 12, 3)
# generates 2, 5, 8, 11

range(-7, -30, -5)
# generates -7, -12, -17, -22, -27

# range(stop) or range(start, stop[, step])
```

BREAK AND CONTINUE

```
for n in range(2, 10):
    if n == 6:
        break
    print(n, end=', ')
# => 2, 3, 4, 5,
# "break" breaks out of the smallest enclosing for or while loop

for letter in "STELLAR":
    if letter in "LE":
        continue
    print(letter, end='*')

# => S*T*A*R*
# continue continues with the next iteration of the loop
```

WHILE LOOPS

```
# Print powers of three below 10000
n = 1
while n < 10000:
    print(n)
    n *= 3
```

(9) Functions

```
# The def keyword defines a function
# Parameters have no explicit types
def fn_name(param1, param2):
   value = do_something()
   return value

#return is optional if either return or its value are
# omitted, implicitly returns None
```

Prime Number Generator

```python
def is_prime(n):
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

n = int(input("Enter a number: "))
for x in range(2, n):
    if is_prime(x):
        print(x, "is prime")
    else:
        print(x, "is not prime")
```

String Redux

Special Characters

```
print('doesn\'t') # => doesn't
print("doesn't") # => doesn't

print('"Yes," he said.') # => "Yes," he said.
print("\"Yes,\" he said.") # => "Yes," he said.

print('"Isn\'t," she said.') # => "Isn't," she said.

#Just choose the easiest string delimiter to
# work with!
```

USEFUL STRING METHODS

```
greeting = "Hello world! "

greeting[4] # => 'o'
'world' in greeting # => True
len(greeting) # => 13

greeting.find('lo') # => 3 (-1 if not found)
greeting.replace('llo', 'y') # => "Hey world!"
greeting.startswith('Hell') # => True
greeting.isalpha() # => False (due to '!')

greeting.lower() # => "hello world! "
greeting.title() # => "Hello World! "
greeting.upper() # => "HELLO WORLD! "

greeting.strip() # => "Hello world!"
greeting.strip('dH !') # => "ello worl"
```

STRINGS ↔ LISTS

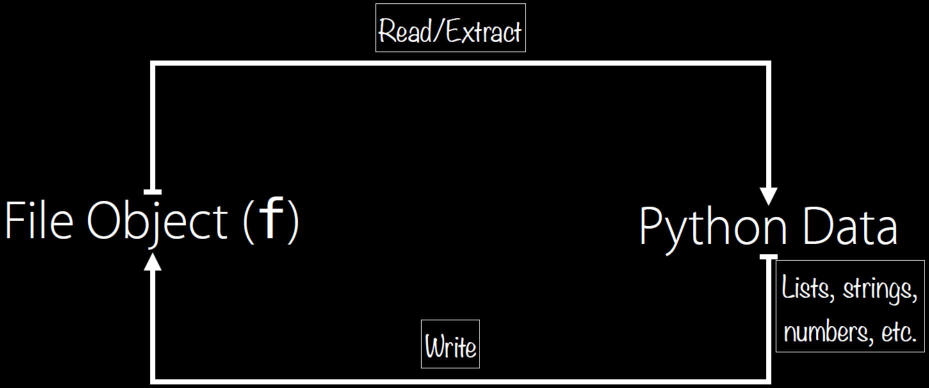
```
# `split` partitions a string by a delimiter
'ham cheese bacon'.split()
# => ['ham', 'cheese', 'bacon']

'03-30-2016'.split(sep='-')
# => ['03', '30', '2016']

# `join` creates a string from a list (of strings)
', '.join(['Eric', 'John', 'Michael'])
# => "Eric, John, Michael"
```

String Formatting

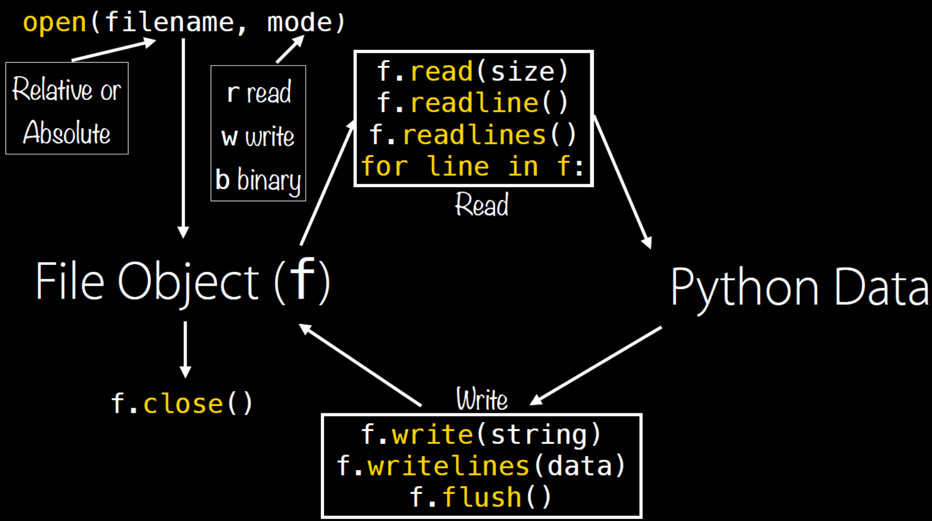
```
# Curly braces in strings are placeholders
'{} {}'.format('monty', 'python') # => 'monty python'

# Provide values by position or by placeholder
"{0} can be {1} {0}s".format("strings", "formatted")
"{name} loves {food}".format(name="Sam", food="plums")

# Pro: Values are converted to strings
"{} squared is {}".format(5, 5 ** 2)
```

File I/O

File I/O diagram



File I/O flow



open(filename, mode)
Relative or Absolute
r read
w write
b binary
f.read(size)
f.readline()
f.readlines()
for line in f:
Read
File Object (f)
Python Data
f.close()
Write
f.write(string)
f.writelines(data)
f.flush()

File Reading

```
# suppose knights.txt tracks a knight's jousting wins and losses:
#  Lancelot 6 0
#  Galahad 7 12
#  Geraint 3 1
#  Mordred 0 0

f = open("knights.txt")
for line in f:
   data = line.split(' ')

   name = data[0]
   wins = int(data[1])
   losses = int(data[2])

   win_percent = 100 * wins / (wins + losses)
   print("%s: Wins %.2f" %(name, win_percent))
f.close()

# something goes wrong here!
# Better ways to unpack the data??
```

USING CONTEXT MANAGERS

```
f = open("file.txt", 'w')
print(1 / 0) # Crash!
f.close()

# The file is never closed! That's bad!

# ALTERNATIVE FOR FILE READING
with open('knights.txt', 'r') as f:
   content = f.read()
   print(1/0)

f.closed # => True

# The with expr as var construct ensures that expr
#  will be "entered" and "exited" regardless
#  of the code block execution

# `content` is still in scope
'content' in locals()
```

Scripts, Modules, Imports

SCRIPTS

- Interactive interpreter:
  - Problem: Temporary
  - Solution: Write code in a file
- First script:

```
#1st line (Shebang) specifies default executable and options

#!/usr/bin/env python3 -tt
""" File: hello.py """

def greet(name):
   print("Hey {}, I am Python!".format(name))

# Run only if called as a script
if __name__ == '__main__':
   name = input("What is your name? ")
   greet(name)

# The special __name__ variable is set to
# '__main__' if your file is executed as a script
```

Running Python Scripts

```
#Supply the filename of the Python script
# to run after the python/python3 command
Lecture_codes\lec1_py> python hello.py
What is your name? John
Hey John, I am Python!

#Supplying the -i option (for 'interactive') will enter the
# interactive interpreter after running the python script
Lecture_codes\lec1_py> python -i hello.py
What is your name? John
Hey John, I am Python!
>>> greet("Jack")
Hey Jack, I am Python!

#Now we have access to symbols from our script.
# Great for debugging!
```

EXECUTABLE SCRIPTS

```
#We can make the Python script executable with chmod,
# as long as the shebang line specified
# a Python interpreter

# This works in linux/mac machines
Lecture_codes\lec1_py> chmod +x hello.py
Lecture_codes\lec1_py> ./hello.py
What is your name? John
Hey John, I am Python!

#in Windows, file is already executable
# (if you open it with python software)
```

USING MODULES

```
#We almost always import the whole
# module, rather than specific symbols

# Import a module
import math
math.sqrt(16) # => 4

# Import specific symbols from a module into the local namespace
from math import ceil, floor
ceil(3.7) # => 4.0
floor(3.7) # => 3.0

# Bind module symbols to a new symbol in the local namespace
from some_module import super_long_symbol_name as short_name

# Any python file (including those you write) is a module
from my_file import my_function, my_variable
```