INTRO TO PYTHON

SECTION 4

Emily Zhang
Senior Software Engineer

Outline

- **Review**
  - string, list, dict
  - common operations on them
- **Set**
  - set.add()
  - set.clear()
  - set.remove()
  - union, difference, intersection
- **Class**
  - brief intro

Review

string list dict

String Review

- **String is a Python sequence, immutable**
- **String & List**

  "+" for concatenation, "*" sign for repetition
- **Indexing & Slicing**

  index: 0, 1, 2, …, n-2, n-1
- **Looping over String**

```
for ch in my_str:
    print(ch)
```

- **String Formatting**

```
"Hello %s and welcome to %s!" % ('John', "GDI")
"Hello {0} and welcome to {1}!".format('John', "GDI")
```

List Review

- **sequential data, allow mixed data type or nested data, mutable**

  my_list = [1, 'hello', [2, 3], {"name": "john'}, None]

- **list/tuple indexing/slicing:** same as string, 0 to len()-1

- **methods**
  - my_list.append(x): add new element to end of the list
  - my_list.pop(): remove last element
  - '<sep>'.join(my_list): form a string from a list, separated by <sep> (this is string method)
  - all list methods: dir(list) or dir(list_var_name)

- **looping**
  - for x in [69, 88, 81, 87, 99, 91, 76, 79]:

    print(x)

Dict Review

★ **Unordered data structure, with key/value pairs, fast look up value by key**

```
grades = {
        "john": { "id": 1111111, "midterm": 98, "final": 96 },
        "kate": { "id": 2222222, "midterm": 86, "final": 77 }
    }
```

★ **Dict Operation**

- add new: my_dict[new_key] = new_value
- updating existing data: my_dict[old_key] = new_value
- delete entry: my_dict.pop(key)
- dict.get(): my_dict.get(key[, default])
- all dict methods: dir(dict) or dir(dict_var_name)

★ **Looping over Dict**

-
```
for k, v in my_dict.items():
    pass
```

Common Operation Review

- **"in" Operator**: string, list, tuple, dict
  - string: check substring existence
  - list, tuple: check element exists
  - dict: check element exists as a key

- **len() built in function**
  - string, list, tuple, dict, set

- **sorted() built in function**, returns new sorted list.
  - list, tuple, string, set
  - `sorted([2, 1, 3])` → [1, 2, 3]
    `sorted([2, 1, 3], reverse=True)` → [3, 2, 1]
    `sorted('hello')` → ['e', 'h', 'l', 'l', 'o']
    `sorted((2, 1, 3))` → [1, 2, 3]

Practice 1: student score list

➔ **Create a list of 100 student scores, that looks similar to this:**

grades = [[student_id1, 98], [student_id2, 76], …..]

➔ **Hint:**

```
import random
random.randint(100000, 999999)      # will create a random student id that's 6 digits
random.randint(60, 100)             # will create a random student score
```

Practice 2: student score dict

- **Given that we have this list:**

  grades = [[student_id1, 98], [student_id2, 76], …..]

- **Convert the list to a dict**

  grades_dict = { student_id1: 98,   student_id2: 76, …. }

Practice 2: Formatted Print

- **Given that we have the dict:** grades_dict = { student_id1: 98, student_id2: 76, …. }
- **Print each student's score like this:**

  Student with id=179146 has score=98

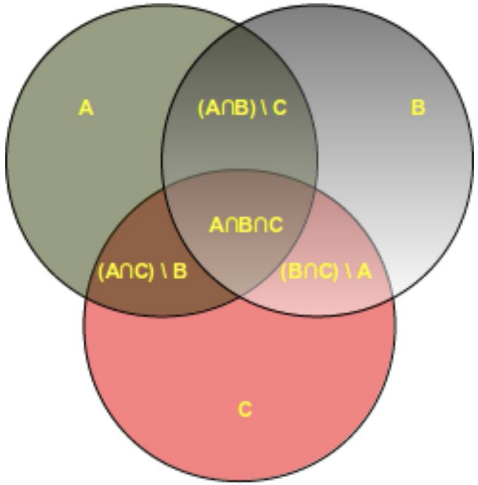
  Student with id=274641 has score=76

Set

What is Set

➔ **List / Tuple / String**

- ordered sequences
- indexed from 0 to n-1 (n being the length)



➔ **Dict**

unordered collection of key/value pairs

➔ **Set**

- unordered collection of keys
- modeled after sets in mathematics.
- example: `>>> x = set(["Postcard", "Radio", "Telegram"]); print(x)`

Set Create

- **Using built-in 'set' function, using another sequence or iterable: list, str, tuple, ...**

```
>>> x = set(["a", "b", "c"])                    # from list
>>> y = set(("a", "b", "c"))                    # from tuple

>>> z = set("abc")                              # from string

>>> set({"name": "a", "height": "183cm"})       # from dict
{"name", "height"}                              # set will store the keys
```

- **Simpler Syntax: set literal**

```
>>> y = {"Postcard","Radio","Telegram"}
>>> print(y)
```

Set Methods

➔ **Clear All Elements**

```
>>> x = set(["Postcard", "Radio", "Telegram"])
>>> x.clear();   print(x)                    # list, dict, set all have clear() method
```

➔ **Add One Element**

```
>>> x.add("Telephone"); print(x)             # similar to list.append(x), but not ordered
```

➔ **Remove One Element**

```
>>> x.remove("Radio"); print(x)
```

In, Not In

- **"In" Operator for "set": check element exists**

```
>>> basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
>>> print(basket)                                  # set has no duplicate, same as dict keys
{'orange', 'banana', 'pear', 'apple'}
```

-
```
>>> 'orange' in basket                             # fast membership testing, O(1) in time
True
>>> 'crabgrass' in basket                          # what data type does 'in' return?
False
>>> 'crabgrass' not in basket
True
```

Practice 4: Set

1. **Convert this sentence to a set of letters**

```
>>> s = "python set is an unordered collection of objects"
>>> (you code here)
>>> my_set
{'o', 's', 'c', 'h', 'j', 'i', 'u', 'd', 'b', ' ', 'p', 'e', 'r', 'n', 'a', 'f', 't', 'y', 'l'}
```

2. **if a vowel letter exist in this set, print it**

```
>>> (you code here)
Letter 'a' is in the set
Letter 'e' is in the set
….
```

3. **Print the length of my_set**

Practice 4: Solution

1. **Convert this sentence to a set of letters**

```
>>> s = "python set is an unordered collection of objects"
>>> my_set = set(s)
>>> my_set
{'o', 's', 'c', 'h', 'j', 'i', 'u', 'd', 'b', ' ', 'p', 'e', 'r', 'n', 'a', 'f', 't', 'y', 'l'}
```

2. **if a vowel letter exist in this set, print it**

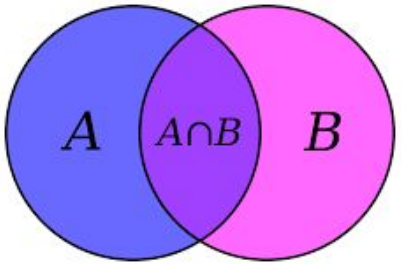
```
>>> for ch in "aeiou":
...         if ch in my_set:
...             print("Letter %s is in the set" % ch)
```

3. **Print the length of my_set:** `>>> len(my_set)`

Union | Difference

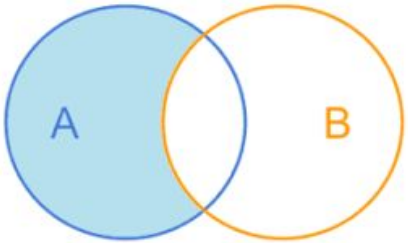
- **Union of 2 sets**

```
>>> a = {1, 3, 5, 7}
>>> b = {1, 2, 4, 6}
>>> a | b                    # or:  a.union(b)
{1, 2, 3, 4, 5, 6, 7}
```



- **Difference between 2 sets**

```
>>> x = {1, 2, 4, 5};  y = {4, 5, 6, 7}
>>> x.difference(y)        # {1, 2},  same as:   x - y
>>> y.difference(x)        # {6, 7},  same as:   y - x
```

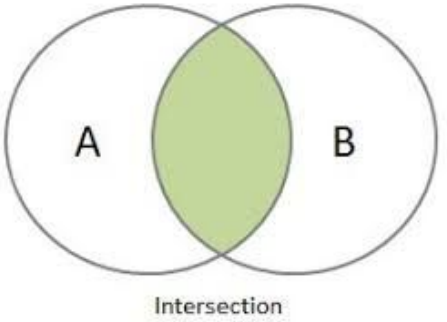


std::set_difference

Intersection

- **Intersection of 2 sets**

```
>>> a = {1, 3, 5, 7};  b = {1, 2, 4, 6}
>>> a & b                          # or:  a.intersection(b)
{1}
```



Practice 5: Set Operations

```
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> ???                          # Q1:  set of unique letters in a
{'a', 'r', 'b', 'c', 'd'}
>>> ???                          # Q2:  set of letters in a but not in b
{'r', 'd', 'b'}
>>> ???                          # Q3:  set of letters in a or b or both
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> ???                          # Q4:  set of letters in both a and b
{'a', 'c'}
```

Practice 5: Solution

```
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                          # Q1:  set of unique letters in a
{'a', 'r', 'b', 'c', 'd'}
>>> a - b                      # Q2:  set of letters in a but not in b
{'r', 'd', 'b'}
>>> a | b                      # Q3:  set of letters in a or b or both
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b                      # Q4:  set of letters in both a and b
{'a', 'c'}
```

Class

Class

- **Data** Collection Types
  - String / List / Tuple / Dict / Set: ways to hold data together
- **Functions**
  - to group some statement as a logical block
- **Class**: a way of grouping together related **data** and **functions**
  - **Encapsulate** the data and functionalities within an object
  - Functions may act upon the data inside of the class, or instance of the class
- **Class is a data type**
  - like: string, list, dict, these are all data types

Class Example

```
# class_basic.py


class Person:

    def __init__(self, name, age=20):
        self.name = name
        self.age = age

    def is_adult(self):
        return self.age > 18
```

- **Class** Keyword
  - to define a class, by convention we use **CamelCase** for class names
- **Colon** after class name
  - similar to if statement followed by colon
  - similar to def function followed by colon
- __init__
  - called when creating an instance of the class, using the passed in arguments
  - often called the 'constructor' or 'initialiser' of a class
- self
  - first parameter of all class's methods
  - refers to the instance of the object
  - `>>> p1 = Person('Tom')`
    `>>> p1.age  #  here "p1" is the "self"`

Class is widely used

https://github.com/pandas-dev/pandas

search 'class', click 'Python' on the left to only select Python code

asv_bench/benchmarks/attrs_caching.py — Python

Showing the top two matches Last indexed on Dec 11, 2017

```
8   from .pandas_vb_common import setup  # noqa
9
10
11  class DataFrameAttributes(object):
12
13      goal_time = 0.2
14
15      def setup(self):
…
23          self.df.index = self.cur_index
24
25
26  class CacheReadonly(object):
27
28      goal_time = 0.2
29
30      def setup(self):
```

Instance Variables

```
# class_basic.py


class Person1:

    def __init__(self, name, age=20):
        self.name = name
        self.age = age

    def is_adult(self):
        return self.age > 18
```

Instance Variable

- **name, age**
- access **instance variable** by
  - instance_name.var
  - **dot notation**

```
>>> from class_basics import Person1
>>> p1 = Person1('Tom', 23)
>>> p1.name                          #   'Tom'
>>> p1.age                           #   23

>>> p2 = Person1('Kate', 30)
>>> p2.name

>>> type(p1)        # <class 'class_basics.Person1'>
```

Class Variables

```
# class_basic.py


class Person2:

    TITLES = ('Dr', 'Mr', 'Mrs', 'Ms')

    def __init__(self, title, name):
        if title not in self.TITLES:
            raise ValueError("%s is invalid." % title)

        self.title = title
        self.name = name
```

Class Variable

- TITLES
- can be accessed in **two** ways
  - Person2.TITLES
  - self.TITLES

```
>>> from class_basics import Person2
>>> Person2.TITLES
('Dr', 'Mr', 'Mrs', 'Ms')

>>> p1 = Person2('Mr', 'Tom')
>>> p1.TITLES
('Dr', 'Mr', 'Mrs', 'Ms')
```

Practice 6: Class

1. Create a class called "Vehicle", that stores the following information
    a. **color, style, price, name**
    b. write a method called "**get_description**" that will return a string based on instance variables, example
        i. Fer is a red convertible worth $60000.00.

2. Create two new vehicles called car1 and car2.
    a. Set car1 to be a red convertible worth $60,000.00 with a name of Fer
    b. Set car2 to be a blue van named Jump worth $15,000.00.

Practice 6: Solution

```
# class_practice.py
 class Vehicle:

   def __init__(self, name="", style="car", color="", value=10000.00):
       self.name = name
       self.style = style
       self.color = color
       self.value = value

   def description(self):
       desc_str = "%s is a %s %s worth $%.2f." % (self.name, self.color, self.style, self.value)
       return desc_str


car1 = Vehicle("Fer", "convertible", "red", 60000.00)
car2 = Vehicle("Jump", "van", "blue", 15000.00)

print(car1.description())
print(car2.description())
```

Summary

- ★ **Scalar Data Type**
  - int, float, boolean, None, str
- ★ **Collection Data Type**
  - sequence: list, tuple
  - dict, set
- ★ **Control Flow**
  - if, elif, else
  - for/while loop, break, continue
- ★ **Built in function**
  - len, print, int, float, str, dict, tuple, set, eval, input, sorted, sum, round, type, help, open
- ★ **Function & Class**

Thank You