CS 314 Lecture 5

February 5, 2019

Python

Strings

```
1 s = '42,13,7,26'
2 s[0]
```

Strings

```
1 s = '42,13,7,26'
2 s.split(',')
```

Strings

```
1 s = '42 13 7 26'
2 s.split()
```

Strings

```
1 s = '42 13 7 26'
2 [int(x) for x in s.split()]
```

Strings

```
1 strs = [ 'bob', 'carol', 'david' ]
2 ';'.join(strs)
```

Loops again

```
1 xs = [ 42, 13, 7, 26 ]
2 for x in xs:
3     print(x)
```

Loops again

```
1 xs = [ 42, 13, 7, 26 ]
2 for i in range(len(xs)):
3     print(f'Element {i} is {xs[i]}')
```

Loops again

```
1 xs = [ 42, 13, 7, 26 ]
2 for i, x in enumerate(xs):
3     print(f'Element {i} is {x}')
```

Dictionaries

```
states = { 'New Jersey': 'NJ',
           'California': 'CA',
           'Texas': 'TX' }

print(states['New Jersey'])
```

Dictionaries

```
states = { 'New Jersey': 'NJ',
           'California': 'CA',
           'Texas': 'TX' }

for k in states:
    print(k)
```

Dictionaries

```
states = { 'New Jersey': 'NJ',
           'California': 'CA',
           'Texas': 'TX' }

for k in states:
    print(f"{k}'s abbreviation is {states[k]}")
```

Dictionaries

```
states = { 'New Jersey': 'NJ',
           'California': 'CA',
           'Texas': 'TX' }

for k, v in states.items():
    print(f"{k}'s abbreviation is {v}")
```

Aside: destructuring assignment

```
1 def foo():
2     return (10, 20)
3
4 p = foo()
5 print(p[0])
6 print(p[1])
```

Aside: destructuring assignment

```
1 def foo():
2     return (10, 20)
3
4 x, y = foo()
5 print(x)
6 print(y)
```

Dictionaries

But this generates an exception (KeyError):

```
states = { 'New Jersey': 'NJ',
           'California': 'CA',
           'Texas': 'TX' }

print(states['New York'])
```

Dictionaries

```
states = { 'New Jersey': 'NJ',
           'California': 'CA',
           'Texas': 'TX' }

try:
    print(states['New York'])
except:
    print("I don't know that state")
```

Dictionaries

```
states = { 'New Jersey': 'NJ',
           'California': 'CA',
           'Texas': 'TX' }

try:
    print(states['New York'])
except KeyError:
    print("I don't know that state")
except Exception:
    print("uh oh")
```

Exceptions

```
1 raise Exception("something bad happened")
```

Subclassing

```
1 class Person:
2     pass
3
4 class Student(Person):
5     pass
```

Subclassing

This doesn't quite work:

```
1 class Person:
2     def __init__(self, name):
3         self.name = name
4
5 class Student(Person):
6     def __init__(self, gpa):
7         self.gpa = gpa
```

Subclassing

This doesn't quite work:

```
1 class Person:
2     def __init__(self, name):
3         self.name = name
4
5 class Student(Person):
6     def __init__(self, name, gpa):
7         Person.__init__(self, name)
8         self.gpa = gpa
```

Subclassing

Python supports multiple inheritance:

```
1  class Employee:
2      def __init__(self, taxID):
3          self.taxID = taxID
4
5  class Student:
6      def __init__(self, gpa):
7          self.gpa = gpa
8
9  class GradStudent(Employee,Student):
10     def __init__(self, taxID, gpa):
11         Employee.__init__(self, taxID)
12         Student.__init__(self, gpa)
```

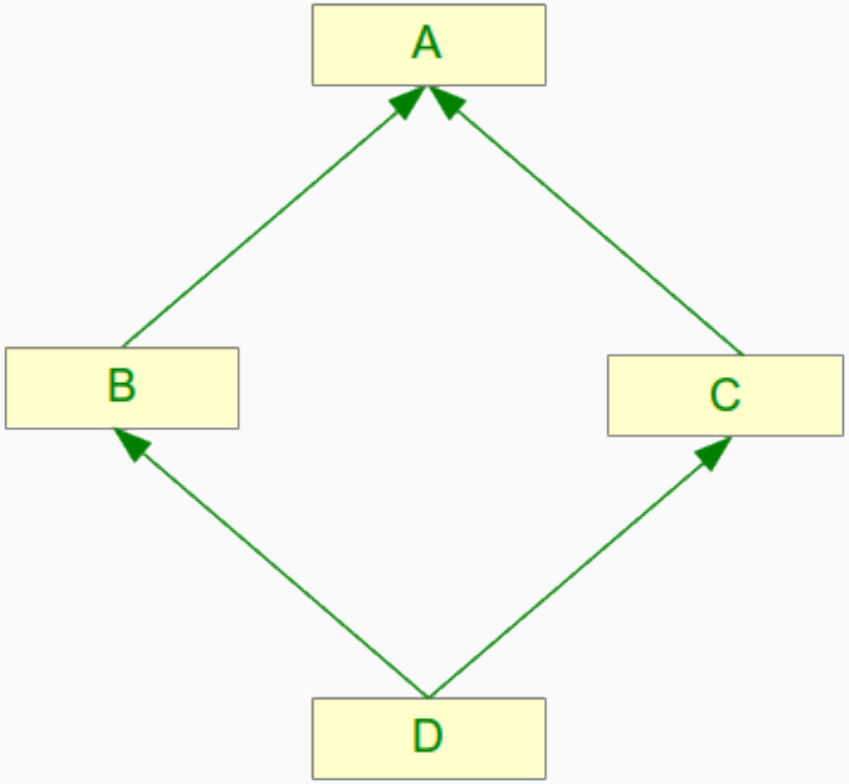
Subclassing

Python supports multiple inheritance:

```
1  class Employee:
2      def __init__(self, taxID):
3          self.taxID = taxID
4
5      def __str__(self):
6          return f'My taxID is {self.taxID}'
7
8  class Student:
9      def __init__(self, gpa):
10         self.gpa = gpa
11
12     def __str__(self):
13         return f'My gpa is {self.gpa}'
```

Multiple inheritance

The “diamond problem”:



Multiple inheritance

```
1  class A:
2      def f(self):
3          print("f in A called")
4
5  class B(A):
6      def f(self):
7          print("f in B called")
8
9  class C(A):
10     def f(self):
11         print("f in C called")
12
13 class D(B,C):
14     pass
```

Multiple inheritance

```
1  class A:
2      def f(self):
3          print("f in A called")
4
5  class B(A):
6      pass
7
8  class C(A):
9      def f(self):
10         print("f in C called")
11
12 class D(B,C):
13     pass
```

Multiple inheritance

```
1 A.f(d)
2 B.f(d)
3 C.f(d)
```

Multiple inheritance

```
class A:
    def f(self):
        return 10 + self.g()

a = A()
a.f()     # error
```

Multiple inheritance

```
1  class A:
2      def f(self):
3          return 10 + self.g()
4
5  class B:
6      def g(self):
7          return 42
8
9  class C(A,B):
10     pass
```

Eval

Python is interpreted, so we can ask it to evaluate strings as code:

```
1 eval('2 + 2')
```

Eval

But eval, especially combined with user input, is dangerous:

```
1 eval('os.listdir(".")')
```

A simple REPL

```
while True:
    s = input('>>> ')
    try:
        exec(f'print(repr({s}))')
    except TypeError:
        exec(s)
```

Optional static typing

```
1  def get_first_name(full_name):
2      return full_name.split(" ")[0]
3
4  defaults = {
5      "name": "Bob Jones",
6      "address": "123 Main St."
7  }
8
9  raw_name = input("Please enter your name: ")
10 first_name = get_first_name(raw_name)
11
12 # If the user didn't type anything in, use the fallback name
13 if not first_name:
14     first_name = get_first_name(defaults)
15
16 print(f"Hi, {first_name}!")
```

(Adapted from https://medium.com/@ageitgey/learn-how-to-use-static-type-checking-in-python-3-6-in-10-minutes-12c86d72677b)

Optional static typing

```
1 x = 42
```

```
1 x: int = 42
```

Optional static typing

```
1  from typing import List,Dict,Tuple
2
3  x: int = 42
4  name: str = "Bob"
5  zips: List[int] = [ 12345, 90210 ]
6  bobUser: Tuple[str, int, str] = ("bob", 500, "/home
       /bob")
7
8  def f(s: str) -> int:
9      return len(s)
10
11 states: Dict[str, str] = { 'New Jersey': 'NJ' }
```

Optional static typing

```
$ mypy typed.py
typed.py:15: error: Unsupported operand types for + ("int" and "str")
typed.py:16: error: Argument 1 to "f" has incompatible type "int"; expected "str"
typed.py:17: error: Argument 1 to "append" of "list" has incompatible type "float"; expected "int"
```

Optional static typing

```
1  from typing import Dict
2
3  def get_first_name(full_name: str) -> str:
4      return full_name.split(" ")[0]
5
6  defaults: Dict[str, str] = {
7      "name": "Bob Jones",
8      "address": "123 Main St."
9  }
10
11 raw_name: str = input("Please enter your name: ")
12 first_name: str = get_first_name(raw_name)
13
14 # If the user didn't type anything in, use the fallback name
15 if not first_name:
16     first_name = get_first_name(defaults)
17
18 print(f"Hi, {first_name}!")
```

Optional static typing

```
$ mypy errTyped.py
errTyped.py:16: error: Argument 1 to "
    get_first_name" has incompatible type "Dict[str
    , str]"; expected "str"
```