CS 314 Lecture 4

January 31, 2019

Dynamic typing

Types prevent some operations:

```
>>> x = 42
>>> x + 'hello'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Dynamic typing

But variables can be reassigned to refer to different types:

```
>>> x = 42
>>> type(x)
<class 'int'>
# ...
>>> x = 'hello'
>>> type(x)
<class 'str'>
```

Classes

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

p1 = Person('Adam', 20)
p2 = Person('Karen', 32)

print(p1.name)
```

Namespaces

```
x = 42
y = 43
z = 44
```

A.py

```
import A

print(A.x)
```

B.py

Duck typing

```python
class Duck:
    def fly(self):
        print('Duck flying')

class Airplane:
    def fly(self):
        print('Airplane flying')

duck = Duck()
airplane = Airplane()

duck.fly()     # prints ‘Duck flying‘
airplane.fly() # prints ‘Airplane flying‘
```

Duck typing

```python
class Duck:
    def fly(self):
        print('Duck flying')

class Airplane:
    def fly(self):
        print('Airplane flying')

def lift_off(entity):
    entity.fly()

duck = Duck()
airplane = Airplane()

lift_off(duck)     # prints ‘Duck flying‘
lift_off(airplane) # prints ‘Airplane flying‘
```

Duck typing

“If it walks like a duck and it quacks like a duck, then it must be a duck”

Anonymous functions

```python
def dbl(x):
    return x * 2

dbl(10)
```

```python
dbl = lambda x: x * 2

dbl(10)
```

Program state

```
x = 42

y = 'hi'

z = 3.5
```

Program state

```
x = 42
# {x: (int, 42)}

y = 'hi'

z = 3.5
```

Program state

```
x = 42
# {x: (int, 42)}

y = 'hi'
# {x: (int, 42), y: (str, 'hi')}

z = 3.5
```

Program state

```
x = 42
# {x: (int, 42)}

y = 'hi'
# {x: (int, 42), y: (str, 'hi')}

z = 3.5
# {x: (int, 42), y: (str, 'hi'), z: (float, 3.5)}
```

Debugging can be tricky

```
my_variable = 10
while my_variable > 0:
    i = foo(my_variable)
    if i < 100:
        my_variable++
    else
        my_varaible = (my_variable + i) / 10
```

Debugging can be tricky

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

p1 = Person('bob', 20, 'student')
p1.ocupation = 'painter'
print(p1.occupation)
```

Debugging can be tricky

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
        self.occupation = 'unemployed'

p1 = Person('bob', 20)
print(p1.occupation)
```

Debugging can be tricky

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
#       self.occupation = 'unemployed'

p1 = Person('bob', 20)
print(p1.occupation)
```

Testing

```python
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

if __name__ == '__main__':
    unittest.main()
```

Overriding default behavior

```python
class Foo:
    def __init__(self):
        self.x = 10

    def __str__(self):
        return 'This object has x = ' + str(self.x)

f = Foo()
print(f)
```

Overriding default behavior

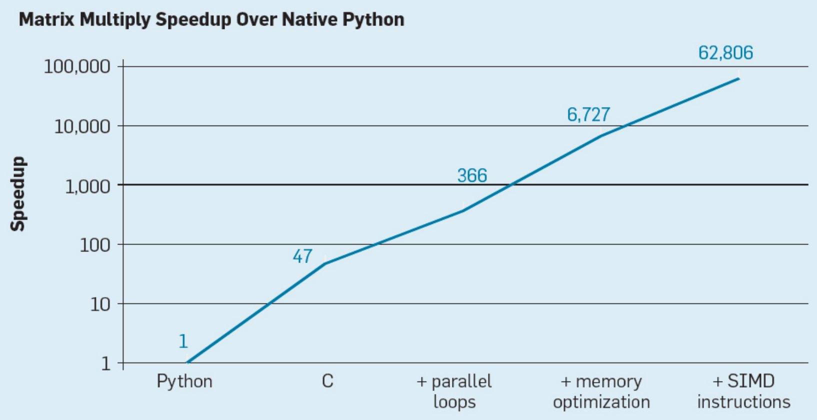
```python
class Attrs:
    def __getattribute__(self, a):
        print(f'Getting {a}')
        return object.__getattribute__(self, a)

    def __setattr__(self, k, v):
        print(f'Setting {k} to {v}')
        object.__setattr__(self, k, v)

attr = Attrs()
attr.x = 10
print(attr.x)
```

Performance

Python can be slow:



CACM: A New Golden Age for Computer Architecture

Performance

One solution: write in both C and Python!

```c
int factorial(int n)
{
    int rv = 1;
    for (int i = 2; i < n; i++)
        rv *= i;
    return rv;
}
```

```python
print(factorial(10))
```

Performance

```
gcc -c foo.c
gcc -shared -o libfoo.so foo.o
```

```python
from ctypes import cdll
lib = cdll.LoadLibrary('./libfoo.so')
print(lib.factorial(10))
```

Interactive environments

- python repl
- IPython / Jupyter