Accelerating Information Technology Innovation

http://aiti.mit.edu

Ghana Summer 2011
Lecture 10 – Becoming a Python Ninja

MISTI


Python

Python Pow

- In encryption, we like to do (a^b)%c
- A, b, and c can be very large numbers.
- Ex: `(1234567890**9876543219)%33`
  - This is very slow. (wasn't done in 3 hours)
  - 650MB of ram, processor maxed out.
- Better way:
  `pow(1234567890,9876543219,33)`
  - At least 1800x faster. (6.14 seconds)
  - The answer is 24.

Reading a text file

- Easy in python:

```
For line in open(“asdf.txt”):
    print line
```

Timing your code

```
import Timer
t = timeit.Timer(“8**2”)
print t.timeit()
```

- If you want to time something longer, use the timer to call a method.

Efficient swapping of variables

- The normal way:

```
c=a
a=b
b=c
```

- The Python way:

```
a,b = b,a
```

  - More efficient – a temporary variable is never created.

Inline Conditionals

- You can do inline if/else statements to make simple coding shorter (similar to the “a ? b : c” concept in other languages)
- Ex:

```
Print “Equal” if A==B else “Not Equal”
```

Sets

- Sets don’t have duplicate values.
- If you only want unique values in a list, you can create a set from it:

```
Print set([1,1,2,2,2,3,3,3,3,4])
```

- Output: set([1,2,3])

Chained comparison operators

- Comparison operators can be chained:

```
X = 5
Return 1<x<10
```

Output: `True`

Step argument for slice operators

```
X = [1,2,3,4,5,6]
```

`Print x[::2]` → [1,3,5]

`Print x[::3]` → [1,4]

`Print x[::-1]` → [6,5,4,3,2,1]

`Print x[::-2]` → [6,4,2]

`Print x[::-2][::-1]` → [2,4,6]

If any, if all

- `numbers = [1,2,3,4,5,6,7]`
- `If any(num for num in numbers)>6`
  - True if any number is greater than 6
- `If all(num for num in numbers)>6`
  - True only if all numbers are greater than 6

List comprehension

- Traditional for loop:

```
X = []
Y = [1,2,3,4,5,6]
for n in y:
    x.append(n**2)
```

- List Comprehension

```
X = [n**2 for n in y]
```

List comprehensions

- They get even better:

```
[n**2 for n in x if n>3]
```

(only if n > 3)

```
[(n,n**2) for n in x]
```

(tuple with n and n^2)

List Comprehensions

- The Normal way:

```
mult_list = []
for a in [1,2,3,4]:
    for b in [5,6,7,8]:
        mult_list.append(a*b)
```

- The Python way:

```
mult_list= [a*b for a in [1,2,3,4]
for b in [5,6,7,8]]
```

Generators

- Generators have the same syntax as list comprehensions, but use parenthesis instead of square brackets
- These are faster than list comprehensions and use much less memory, but can’t store your data.
- Computes one value at a time.

Generators

- List comprehension
  - sum([a^b for a in range(1000) for b in range(1000)])
  - The complete list comprehension is created first, stored in memory, and summed after completion.
  - 25 seconds, >600MB ram
- Generator
  - sum(a^b for a in range(1000) for b in range(1000))
  - Values are added to the sum one at a time
  - 23 seconds, <0.5MB ram

Lambda functions

- A function that is created at runtime.
- Always returns something (but doesn't include a return statement)
- Convenient for passing as an argument
- Ex:

```
f = lambda x:x**2
```

- Takes x as input and returns x^2

Lambda vs function:

Lambda:

```
f = lambda x:x**2
```

Function:

```
def square(x):
    return x**2
```

Filter Function

- Syntax: filter(function, list)
- Ex:

```
numbers = [1,2,3,4,5,6,7]
print filter(lambda x: x<4, numbers)
```

Output:

```
[1,2,3]
```