Python Applications

Performance, debugging

CS101 Lecture #20

2016-12-02

Administrivia

Administrivia

- Timeline
  - Last lecture on Python (lec01-20)
  - Midterm II on Tuesday
  - Start Matlab on Thursday (lec21-25)

- Final exam
  - Dec 29th, 9:00-12:00 (East/West Lecture Hall 102)

Optimization strategy refresh

- Brute-force search
  - Exhaustive search over the entire space
- Hill-climbing
  - Iterative process
  - Greedily improving; Local optimum
- Random walk
  - Iterative process
  - Allow down-hilling steps
- Random sampling
  - random exploration of the search space
  - Non-iterative process

Code Performance

Code performance

- Measure run time
- Performance analysis

revisit the Fibonacci example

```
def fibo_a(n):
    if n == 1 or n == 2:
        return 1
    else:
        return fibo_a(n-1) + fibo_a(n-2)

def fibo_b(n):
    if n == 1 or n == 2:
        return 1
    a, b = 1,1
    for i in range(3,n+1):
        a,b = b, a+b
    return b

def fibo_c(n):
    p = (1+ 5**0.5)/2
    q = 1-p
    return int((p**n - q**n)/5**0.5  + 0.5)
```

$$F_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

Measure the runtime

- Use the `timeit` module
  - Interpreter: `timeit.timeit('stmt', number = 100)`
  - Command line: `python -m timeit -n 100 'stmt'`
  - Notebook: `%timeit -n 100 func(n)` (this is easiest)

Measure the runtime

```
Import timeit
%timeit -n 1000 fibo_a(10)
%timeit -n 1000 fibo_a(20)
%timeit -n 1000 fibo_a(30)
%timeit -n 1000 fibo_a(50)      ##DON'T do!

%timeit -n 1000 fibo_b(10)
%timeit -n 1000 fibo_b(100)
%timeit -n 1000 fibo_b(1000)


%timeit -n 1000 fibo_c(10)
%timeit -n 1000 fibo_c(100)
%timeit -n 1000 fibo_c(1000)
```

Time complexity analysis

```
def fibo_a(n):
    if n == 1 or n == 2:
        return 1
    else:
        return fibo_a(n-1) + fibo_a(n-2)
```

Time = $O(2^n)$

```
def fibo_b(n):
    if n == 1 or n == 2:
        return 1
    a,b = 1,1
    for i in range(3, n+1):
        a,b = b,a+b
    return b
```

Time = $O(n)$

```
def fibo_c(n):
    p = (1 + 5**0.5)/2
    q = 1-p
    return int((p**n - q**n)/5**0.5  + 0.5)
```

Time = $O(1)$

When Things Go Wrong

- Errors are good
  - Errors reveal the boundary of what you know/don't know

Debugging

- A few definitions
  - Exceptions: unusual behaviors occurred in the execution of a program; caught by the try:{…}except e:{…} syntax
  - Errors: exceptions that cause the program to be unrunnable
  - Bugs: errors and exceptions; can also be miswritten, ambiguous, or incorrect code which is exception free and does not advertise its discrepancy

Common exceptions

- SyntaxError
- NameError
- TypeError
- ValueError
- IOError
- IndexError
- KeyError
- ZeroDivisionError
- IndentationError
- Exception

Common exceptions

- SyntaxError: missing comma or parentheses
- NameError: undefined variable or function names
- TypeError: check variable types (coerce if necessary)
- ValueError: built-in functions have valid type of arguments, but invalid values specified
- IOError: File not exist
- IndexError: index out of range for list, tuple, array

Common exceptions

- KeyError: similar to IndexError, but for dictionary
- ZeroDivisionError: 1/0
- IndentationError: indentation not specified properly (python unique)
- Exception: the most generic type; subsumes all the above exceptions and many others

More Exception types:
http://www.tutorialspoint.com/python/python_exceptions.htm

Examples

```
>>> l = [1 2 3]

>>> print(l)

>>> '%i'%'5'

>>> a = int('a')

>>> for i in range(5)
       Print('%s'%i)

>>> a = 1/0.0
```

Catch an exception

```
try:
    a = 1/0
except ZeroDivisionError:
    print('division by zero')
```

Catch an exception

```
try:
    Print('%i'%'5')
except NameError:
    print('Name error occurred!')
except TypeError:
    print('Type error occurred!')
except Exception:
    print('some other errors/exceptions occurred')
else:
    print('all good')
```

Catch an exception

```
try:
    print('%i'%'5')
except NameError:
    print('Name error occurred!')
except TypeError:
    print('Type error occurred!')
except Exception:
    print('some other errors/exceptions occurred')
else:
    print('all good')
```

Catch an exception

```
try:
    Print('%i'%'5')
except ValueError:
    print('Name error occurred!')
except IOError:
    print('Type error occurred!')
```

Catch an exception

```
try:
    Print('%i'%'5')
except ValueError:
    print('Name error occurred!')
except IOError:
    print('Type error occurred!')
except Exception:
    print('some other errors/exceptions occurred')
```

Catch an exception

```
try:
    print('%i'%5)
except Exception:
    print('some other errors/exceptions occurred')
else:
    print('all good')
```

It is always a good programming/software engineering practice to enclose a block of sensitive code with
`try: … except Exception: …`
to catch and stop unusual behaviors under control

Question

```
# calculate squares
d = list(range(10))
while i < 10:
    d[i] = d[i]**2.0
    i += 1
```

Which error would this code produce?

A Syntax Error
B IndexError
C ValueError
D NameError

Question

Which of the following code would produce `TypeError`?

A `'2' + 2`

B `2/0`

C `2e8 + (1+0j)`

D `'2'*2`

Course Overview

- Python basics
  - Operators, expressions, data types, flow controls
- Python applications
  - Workflow, I/O, performance analysis & debugging
- Numerical Python
  - Simulation, modeling, randomization, plotting, optimization ★
- MATLAB: to come

★ (we are here)

Style

Style

- Use descriptive variable name
  - Reserve i,j,k for indices (s for tmp string, c for tmp char)
- Keep consistent naming conventions
  - E.g. Variable: `age, year_sale, price_in_may`
  - E.g. Function name: `GetKey(), CalculateValue()`
- Explicitness is good
  - Use parentheses even unnecessary
  - Leave proper space in expressions
- Write comments!

Style

- Comments

```
x_vals = [0, 0.1, 0.2, 0.3]    # meters
faraday = 9723.2333            # coulombs
                               # electric charge
```

```
def Warning(msg):
    '''Display a warning message.'''
    print('Warning: %s'%msg)
```

- Docstring explaining what the function does and what its parameters are.
- In triple-quoted strings following immediately after the function header

```
>> help(Warning)
```

Style

- What makes a good (Python) code?

```
import this
```