2016-04-04

William A. Stein

4/6/2016

Contents

1 Math 480: Open Source Mathematical Software

1.0.1 2016-04-04

1.0.2 William Stein

1.1 Lectures 4: Basic Python (part 1/3)

Organization:

- hmwk from last week graded and returned
- hmwk for this week available now
- office hours ta discussion
- I will leave immediately after class to do a basic demo of SMC for another class today
- stickers
- remind me to turn on the screencast, since I definitely forgot!

Key things to learn this week:

1. Basics: functions, whitespace is significant, if, while, for, list comprehensions

2. Data structures: lists, tuples, dicts, sets
3. (maybe) Classes: class, methods, inheritence

1.2 Functions

```
def name_of_function(argument1, argument2):
    # This is a function. I changed
    print "the first argument is ", argument1, " and the second is "\
   , argument2
    return argument1 + argument2
```

```
output = name_of_function(15, 19)
```
```
the first argument is  15  and the second is  19
```

```
True
```
```
True
```

```
False
```
```
False
```

```
None
```

```
print output
```
```
34
```

```
name_of_function('math', '480')
```
```
the first argument is  math  and the second is  480
'math480'
```

```
# This should fail -- you can't add a string and a number in  Python\
   .
name_of_function('abc', 123)
```
```
the first argument is  abc  and the second is  123
Error in lines 2-2
Traceback (most recent call last):
  File ‘‘/projects/sage/sage-6.10/local/lib/python2.7/site-
packages/smc_sagews/sage_server.py’’, line 904, in execute
    exec compile(block+'\n', '', 'single') in namespace, locals
  File ‘‘’’, line 1, in <module>
  File ‘‘’’, line 4, in name_of_function
  File ‘‘sage/structure/element.pyx’’, line 1651, in
sage.structure.element.RingElement.__add__
(/projects/sage/sage-6.10/src/build/cythonized/sage/structure/element.c:15852)
    return coercion_model.bin_op(left, right, add)
```

```
  File ‘‘sage/structure/coerce.pyx’’, line 1069, in
sage.structure.coerce.CoercionModel_cache_maps.bin_op
(/projects/sage/sage-6.10/src/build/cythonized/sage/structure/coerce.c:9736)
    raise TypeError(arith_error_message(x,y,op))
TypeError: unsupported operand parent(s) for '+': '<type 'str'>' and 'Integer Ring'
```

```
%python
name_of_function('abc', 123)
```

```
the first argument is  abc  and the second is  123
Error in lines 1-1
Traceback (most recent call last):
  File ‘‘/projects/sage/sage-6.10/local/lib/python2.7/site-
packages/smc_sagews/sage_server.py’’, line 904, in execute
    exec compile(block+'\n', '', 'single') in namespace, locals
  File ‘‘’’, line 1, in <module>
  File ‘‘’’, line 4, in name_of_function
TypeError: cannot concatenate 'str' and 'int' objects
```

Exercise now: Write a function avg that takes four arguments (which you may assume are numbers) and returns their average.

```
def avg(x1, x2, x3, x4):
    # write code here.
```

```
# test code here (etc.)
avg(1,2,3,4)
```

1.3 If/else Statements

if [condition]: [some code] elif [condition]: [some code] ... else: [some code]

The condition can use ==, !=, and, or, not, and so on.

(Note: Python has no switch statement.)

```
def parity(n):
    if n % 2 == 0:
        print "even"
    elif n == 2017:
        print "odd, that is next year!"
    else:
        print "odd"
```

```
parity(3)
```

```
odd
```

```
parity(6)
```

```
even
```

```
parity(2017)
```
```
odd, that is next year!
```

Exercise now: Write a function that takes a number as input and prints "negative" if it is less than 0, "positive" if it is bigger than 0, and "zero" if it is equal to 0.

```
def my_sign(n):
    # write your code here
```

1.4 For loops

'range(n)' is the list of integers from 0 up to n-1.
for [variable name] in [list or anything "iterable"]: [code...]

```
range(10)
```
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
for i in range(10):
    print "Hello", i
```
```
Hello 0
Hello 1
Hello 2
Hello 3
Hello 4
Hello 5
Hello 6
Hello 7
Hello 8
Hello 9
```

```
i
```
```
9
```

```
def sum_up_to(n):
    s = 0
    for i in range(n+1):
        s = s + i
    return s
```

```
sum_up_to(3)
```
```
6
```

```
sum_up_to(10)
```
```
55
```

```
sum_up_to(100)
```
```
5050
```

```
range(7, 50, 3)
```
```
[7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46, 49]
```

```
range?
```

```
Docstring :
range(stop) -> list of integers range(start, stop[, step]) -> list \
   of
integers

Return a list containing an arithmetic progression of integers.
range(i, j) returns [i, i+1, i+2, ..., j-1]; start (!) defaults to \
   0.
When step is given, it specifies the increment (or decrement). For
example, range(4) returns [0, 1, 2, 3].  The end point is omitted!
These are exactly the valid indices for a list of 4 elements.
```

```
[5..15]
```
```
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

```
[5, 5.7, .., 15]
```
```
[5.00000000000000, 5.70000000000000, 6.40000000000000, 7.10000000000000, 7.80000000000000,
8.50000000000000, 9.20000000000000, 9.90000000000000, 10.6000000000000, 11.3000000000000,
12.0000000000000, 12.7000000000000, 13.4000000000000, 14.1000000000000, 14.8000000000000]
```

Exercise now: Write a function sum_pow_up_to(n, k) that returns the sum $1^k + 2^k + \cdots + n^k$. You may assume that n and k are positive integers.

```
4^5
```
```
1024
```

```
4**5
```
```
1024
```

```
%python
4^5
```
```
1
```

```
def sum_pow_up_to(n, k):
    # put code here...
```

```
sum_pow_up_to(100,2)
```

1.5 While loops

while [condition]: [code...]

(You can put break in the code to immediately exit the while loop.)

```
i = 5
while i > 0:
    print i
    i -= 1
```

```
5
4
3
2
1
```

```
def my_factor(n):
    while not is_prime(n):        # is_prime is part of Sage (not \
   Python)
        k = n.trial_division()    # trial_division is part of Sage (\
   not Python)
        n = n//k
        print "found a factor=", k
    print "finished; prime number remaining=", n
```

```
my_factor(20)
```

```
found a factor= 2
found a factor= 2
finished; prime number remaining= 5
```

```
my_factor(2016)
```

```
found a factor= 2
found a factor= 2
found a factor= 2
found a factor= 2
found a factor= 2
found a factor= 3
found a factor= 3
finished; prime number remaining= 7
```

Exercise now:

Use a while loop to write a function largest_power_of_2 that takes an input a positive integer n and returns the largest power of 2 that is less than n. Your loop should start with pow=1 and while pow*2 is less than n replaces pow by pow*2. When the condition fails, just return pow.

```
def largest_power_of_w(n):
    # put code here...
```