Python language: Data structures and functions

The FOSSEE Group

Department of Aerospace Engineering
IIT Bombay

26 September, 2010
Day 2, Session 3

Outline

Outline

Outline

while

Example: Fibonacci series

Sum of previous two elements defines the next

```
In []: a, b = 0, 1
In []: while b < 10:
   ...:     print b,
   ...:     a, b = b, a + b
   ...:
   ...:
1 1 2 3 5 8
```

range()

range([start,] stop[, step])

- **range()** returns a list of integers
- The **start** and the **step** arguments are optional
- **stop** is not included in the list

Documentation convention

- Anything within **[]** is optional
- Nothing to do with Python.

for ...range()

Example: print squares of first 5 numbers

```
In []: for i in range(5):
  ....:     print i, i * i
  ....:
  ....:
0 0
1 1
2 4
3 9
4 16
```

for ...range()

Example: print squares of odd numbers from 3 to 9

```
In []: for i in range(3, 10, 2):
  ....:     print i, i * i
  ....:
  ....:
3 9
5 25
7 49
9 81
```

5 m

Outline

Problem set 1: Problem 1.1

Write a program that displays all three digit numbers that are equal to the sum of the cubes of their digits.
That is, print numbers *abc* that have the property
$abc = a^3 + b^3 + c^3$
For example, $153 = 1^3 + 5^3 + 3^3$

These are called *Armstrong* numbers.

Problem 1.2 - Collatz sequence

1. Start with an arbitrary (positive) integer.
2. If the number is even, divide by 2; if the number is odd, multiply by 3 and add 1.
3. Repeat the procedure with the new number.
4. It appears that for all starting values there is a cycle of 4, 2, 1 at which the procedure loops.

Write a program that accepts the starting value and prints out the Collatz sequence. 10 m

Outline

Outline

Lists

We already know that

```
num = [1, 2, 3, 4]
```

is a list

Lists: methods

```
In []: num = [9, 8, 2, 3, 7]

In []: num + [4, 5, 6]
Out[]: [9, 8, 2, 3, 7, 4, 5, 6]

In []: num.append([4, 5, 6])

In []: num
Out[]: [9, 8, 2, 3, 7, [4, 5, 6]]
```

Lists: methods

```
In []: num = [9, 8, 2, 3, 7]

In []: num.extend([4, 5, 6])
In []: num
Out[]: [9, 8, 2, 3, 7, 4, 5, 6]

In []: num.reverse()
In []: num
Out[]: [6, 5, 4, 7, 3, 2, 8, 9]

In []: num.remove(6)
In []: num
```

Lists: slicing

- `list[initial:final]`

```
In []: a = [1, 2, 3, 4, 5]

In []: a[1:3]
Out[]: [2, 3]

In []: a[1:-1]
Out[]: [2, 3, 4]

In []: a[:3]
Out[]: [1, 2, 3]
```

Lists: slicing

- `list[initial:final:step]`

```
In []: a[1:-1:2]
Out[]: [2, 4]

In []: a[::2]
Out[]: [1, 3, 5]

In []: a[-1::-1]
Out[]: [5, 4, 3, 2, 1]
```

List containership

Recall `num` is `[9, 8, 2, 3, 7]`

```
In [ ]: 4 in num
Out[ ]: False

In [ ]: b = 8
In [ ]: b in num
Out[ ]: True

In [ ]: b not in num
Out[ ]: False
```

Outline

Tuples: Immutable lists

```
In []: t = (1, 2, 3, 4, 5, 6, 7, 8)

In []: t[0] + t[3] + t[-1]
Out[]: 13

In []: t[4] = 7
```

Note:

- Tuples are immutable - cannot be changed

20 m

Tuples: Immutable lists

```
In []: t = (1, 2, 3, 4, 5, 6, 7, 8)

In []: t[0] + t[3] + t[-1]
Out[]: 13

In []: t[4] = 7
```

Note:

- Tuples are immutable - cannot be changed

30 m

A classic problem

Interchange values

How to interchange values of two variables?

Note:

This Python idiom works for all types of variables.
They need not be of the same type!

A classic problem

Interchange values

How to interchange values of two variables?

Note:

This Python idiom works for all types of variables.
They need not be of the same type!

Outline

Dictionaries: recall

```
In []: player = {'Mat': 134,'Inn': 233,
                 'Runs': 10823, 'Avg': 52.53}

In []: player['Avg']
Out[]: 52.530000000000001
```

Note!

Duplicate keys $\Rightarrow$ overwritten!

You can iterate through a dictionary using keys.

Dictionaries: containership

```
In [ ]: 'Inn' in player
Out[ ]: True

In [ ]: 'Econ' in player
Out[ ]: False
```

Note

- We can check for the containership of keys only
- Not values

Dictionaries: methods

```
In [ ]: player.keys()
Out[ ]: ['Runs', 'Inn', 'Avg', 'Mat']

In [ ]: player.values()
Out[ ]: [10823, 233,
         52.530000000000001, 134]
```

Problem Set 2.1: Problem 2.1.1

You are given date strings of the form “29 Jul, 2009”, or “4 January 2008”. In other words a number, a string and another number, with a comma sometimes separating the items.
Write a program that takes such a string as input and prints a tuple (yyyy, mm, dd) where all three elements are ints.

Outline

Sets

- Simplest container, mutable
- No ordering, no duplicates
- usual suspects: union, intersection, subset . . .
- >, >=, <, <=, in, . . .

```
In []: f10 = set([1,2,3,5,8])

In []: p10 = set([2,3,5,7])

In []: f10 | p10
Out[]: set([1, 2, 3, 5, 7, 8])
```

Set . . .

```
In []: f10 & p10
Out[]: set([2, 3, 5])

In []: f10 - p10
Out[]: set([1, 8])

In []: p10 - f10, f10 ^ p10
Out[]: (set([7]), set([1, 7, 8]))

In []: set([2,3]) < p10
Out[]: True
```

Set . . .

```
In [ ]: set([2,3]) <= p10
Out[ ]: True

In [ ]: 2 in p10
Out[ ]: True

In [ ]: 4 in p10
Out[ ]: False

In [ ]: len(f10)
Out[ ]: 5
```

Problem set 2.2: Problem 2.2.1

Given a dictionary of the names of students and their marks, identify how many duplicate marks are there? and what are these?

Problem 2.2.2

Given a list of words, find all the anagrams in the list.

Outline

Functions

- `def` - keyword to define a function
- Arguments are local to a function
- Functions can return multiple values

Functions: example

```
def signum( r ):
    """returns 0 if r is zero
    -1 if r is negative
    +1 if r is positive"""
    if r < 0:
        return -1
    elif r > 0:
        return 1
    else:
        return 0
```

Note docstrings

What does this function do?

```
def what( n ):
    if n < 0: n = -n
    while n > 0:
        if n % 2 == 1:
            return False
        n /= 10
    return True
```

What does this function do?

```
def what( n ):
    i = 1
    while i * i < n:
        i += 1
    return i * i == n, i
```

What does this function do?

```
def what( x, n ):
    if n < 0:
        n = -n
        x = 1.0 / x

    z = 1.0
    while n > 0:
        if n % 2 == 1:
            z *= x
        x *= x
        n /= 2
    return z
```

What did we learn?

- Loops: `while` , `for`
- Advanced Data structures:
  - Lists
  - Tuples
  - Dictionaries
  - Sets
- Functions
- Docstrings