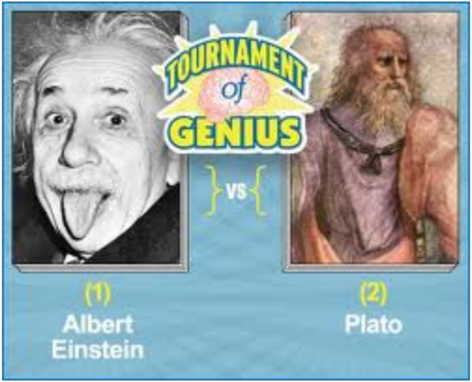
Loop Statements

Using loop statements and graphs

D.S. Hwang

Department of Software Science
Dankook University



The beginning is the most important part of the work.

Plato 427-347BC

Outline

Overview

- Counted Loops
- while loop
- User Input Loops
- Controlling Loops
- Style Notes
- Summary

Counted Loops

Repeat a block.

```
for var in list:
    block

for c in 'alpha':
    print(c )
```

```
while condition:
    block

c = 'alpha'; i=0
while i < len(c):
    print c; i=i+1;
```

Ranges of numbers

A built-in function called `range` generates a list of numbers.

- `range(stop)`
- `range(start,stop)`
- `range(start,stop,step)`

i:j = i, i+1, ...,j-1

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

np.arange(...)

The enumerate Function

Given an iterable object(sequence, list, tuple, a string), `enumerate` returns a list of pairs:- `(index, value)`

- `enumerate(list)`
- `enumerate(sequence)`
- `enumerate(tuple)`
- `enumerate(string)`

(0,'a'),(1,'b'), (2,'c')

```
>>> for x in enumerate('abc'):
        print(x)

>>> for x in enumerate([1, 2, 3, 4, 5]):
        print(x)
```

The enumerate Function

Python allows multivalued assignment.

```
values = [1,2,3]
for pair in enumerate(values):
    idx = pair[0]; val = pair[1];
    values[i] = 2 * val;
```

```
values = [1,2,3]
for (idx,val) in enumerate(values):
    values[i] = 2 * val;
```

while Loops

Repeat a block with a condition.

```
while condition:
    block

c = 'alpha'; i=0
while i < len(c):
    print(c); i=i+1;
```

```
for var in list:
    block

for c in 'alpha':
    print(c)
```

while Loops

Calculate the growth of a bacterial colony using a simple exponential growth model, which is essentially a calculation of compound interest:

$$\boxed{P(t+1) = P(t) + rP(t)}$$

- $P(t)$ is the population size at time t
- r is the growth rate.

while Loops

```
time = 0
population = 1000
# 1000 bacteria to start with
growth_rate = 0.21 # 21% growth per minute
while population < 2000:
  population = population + growth_rate * population
  print(population)
  time = time + 1
  print("It took %d minutes for the bacteria to double."
        % time)
  print("...and the final population was %6.2f bacteria."
        % population)
```

while Loops

```
1210.0
It took 1 minutes for the bacteria to double.
...and the final population was 1210.00 bacteria.
1464.1
It took 2 minutes for the bacteria to double.
...and the final population was 1464.10 bacteria.
1771.561
It took 3 minutes for the bacteria to double.
...and the final population was 1771.56 bacteria.
2143.58881
It took 4 minutes for the bacteria to double.
...and the final population was 2143.59 bacteria.
```

Plot Graphs

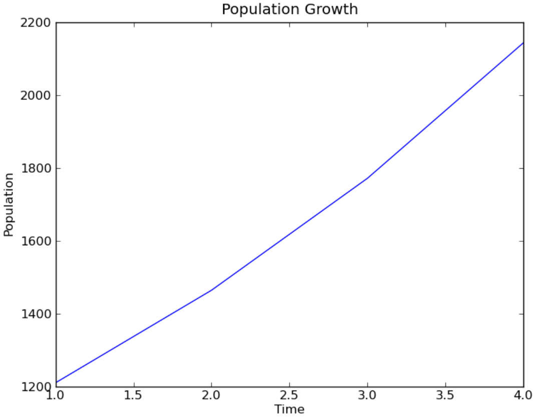
Plot a graph within Python

- install `matplotlib`
  - The Enthought Python Distribution (EPD) for Windows, OS X or Redhat
  - Python (x, y) for Windows
  - `matplotlib` is packaged for pretty much every major linux distribution.
- The main packages are `matplotlib` and `pylab`.
- http://matplotlib.sourceforge.net

Plot Graphs

```
from pylab import *
time = 0
population = 1000
# 1000 bacteria to start with
growth_rate = 0.21 # 21% growth per minute
values = [ ];
times = [ ];
while population < 2000:
  population = population + growth_rate * population
  print population
  time = time + 1
  print("It took %d minutes for the bacteria to double." % time)
  print("...and the final population was %6.2f bacteria." % population)
  values.append(population);
  times.append(time);

plot(times, values);
title('Population Growth');
xlabel('Time'); ylabel('Population');
show()
```

Plot Graphs


Population Growth
2200
2000
1800
1600
1400
1200
Population
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Time

Infinite Loops

What would happen if we stopped only when the population was exactly double its initial size?

```
# Use multi-valued assignment to set up controls.
time, population, growth_rate = 0, 1000, 0.21

# Don't stop until we're exactly double original size.
while population != 2000:
        population = population + growth_rate * population
        print(population)
        time = time + 1
        print("It took %d minutes for the bacteria to double." % time)
```

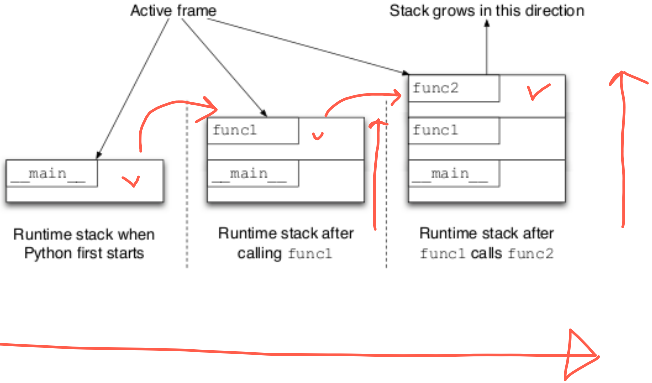
Infinite Loops

Terminate the program.

- Restart Shell from the Options menu in Wing101
- pressing Ctrl-C from the command-line shell

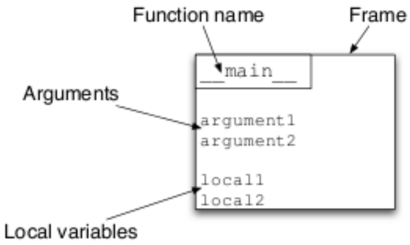
The Call Stack

Python keeps track of any running functions using a runtime stack.



The Call Stack

The stack is a frame that is a series of records. Only the top frame is active; the rest are paused, waiting until functions above them are finished.



The Call Stack

When a function is called, Python

1. creates a new frame for it
2. adds the frame to the top of the stack

On a frame:

- Frames store information about each function call and the order they were called in.
- The most recently called function's frame always sits at the top of the stack.

The Call Stack

- Each stack frame stores a function's parameters and local variables.
- It also contains a reference to the next statement(return address) Python will execute when the function finishes.
- The frame has space set aside for storing the function's return value.
- When your Python program is finished executing, there will be no more frames on the stack.

User input loops

Use the `input` function in a loop to make the chemical formula translation example

```
text = ""
while text != "quit":
  text = input("Please enter a chemical formula
               (or ’quit’ to exit): ")
  if text == "quit":
     print ("...exiting program")
  elif text == "H2O":
     print("Water")
  elif text == "NH3":
     print("Ammonia")
  elif text == "CH3":
     print("Methane")
  else:
     print("Unknown compound")
```

break vs. continue

Two ways of controlling the iteration of a loop

`break` exits the loop body immediately

`continue` skips ahead to the next iteration

```
earth_line = 1
file = open("data.txt", "r")
for line in file:
    line = line.strip()
    if line == "Earth":
        break
    earth_line = earth_line + 1
print("Earth is at line %d" % earth_line)
```

break vs. continue

Two ways of controlling the iteration of a loop

`break` exits the loop body immediately

`continue` skips ahead to the next iteration

```
entry_number = 1
file = open("data.txt" , "r" )
for line in file :
    line = line.strip()
    if line.startswith("#" ):
        continue
    if line == "Earth" :
        break
    entry_number = entry_number + 1
print("Earth is the %dth-lightest planet." % (entry_number))
```

break vs. continue

File data.txt

`break` exits the loop body immediately

`continue` skips ahead to the next iteration

```
# Pluto is only 0.002 times the mass of Earth.
Pluto
Mercury
# Mars is half Earth's diameter, but only
# 0.11 times Earth's mass.
Mars
Venus
Earth
Uranus
```

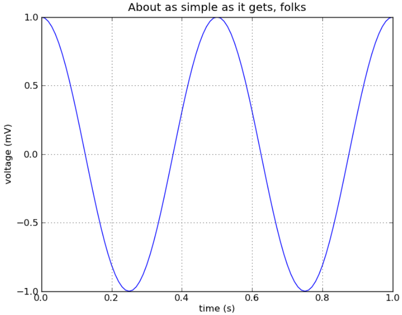
Style Notes

- Reducing the amount of nesting is one way to improve the readability of the code.
- `break` and `continue` have their place but should be used sparingly since they can make programs harder to understand.
- Well-chosen loop conditions can replace `break`, and `if` statements can be used to skip statements instead of `continue`.

Graph

```
# fn: voltage.py
import numpy
import pylab

t = numpy.arange(0.0, 1.0+0.01, 0.01)
s = numpy.cos(2*2*numpy.pi*t)
pylab.plot(t, s)
pylab.xlabel('time (s)')
pylab.ylabel('voltage (mV)')
pylab.title('About as simple as it gets, folks')
pylab.grid(True)
pylab.savefig('simple_plot')
pylab.show()
```

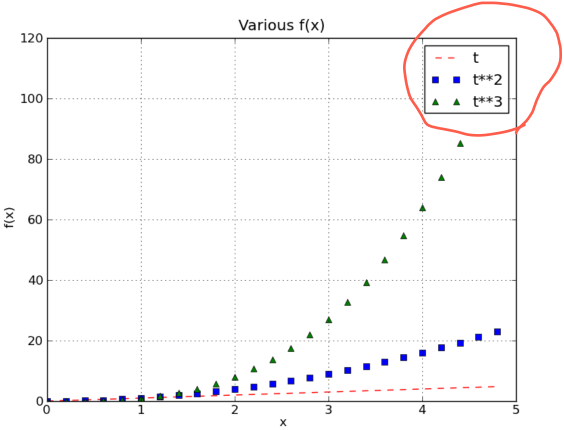
Graph


About as simple as it gets, folks
voltage (mV)
time (s)
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0

Graph

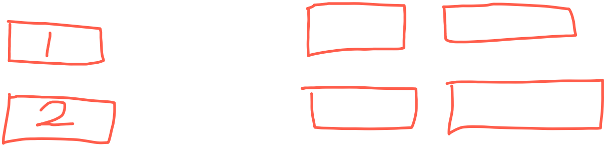
```python
# fn: scatter.py
import numpy as np
import pylab

t = np.arange( 0, 5, 0.2)

pylab.plot(t, t, 'r--', t, t**2,'bs', t, t**3, 'g^')
pylab.xlabel(' x ')
pylab.ylabel(' f(x) ')
pylab.title('Various f(x)')
pylab.grid(True)
pylab.legend(('t', 't**2', 't**3'))

pylab.show()
pylab.savefig('../scatter.png')
```

Graph

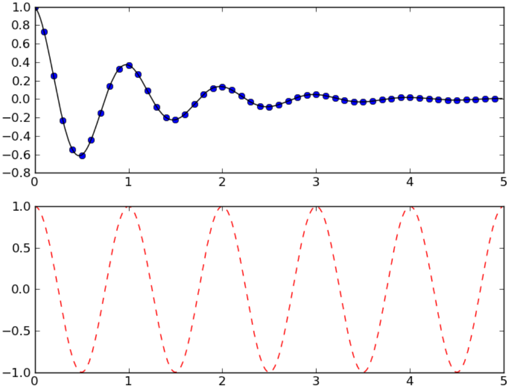


Graph

```python
# fn: multiplot.py
import numpy as np
import pylab as plt

def f(t):
  return np.exp(-t) * np.cos(2*np.pi*t);

t1 = np.arange( 0, 5, 0.1); t2 = np.arange( 0, 5, 0.02)

plt.figure(1);
plt.subplot(211);
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k');

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--');
plt.show()
```



Graph



Graph

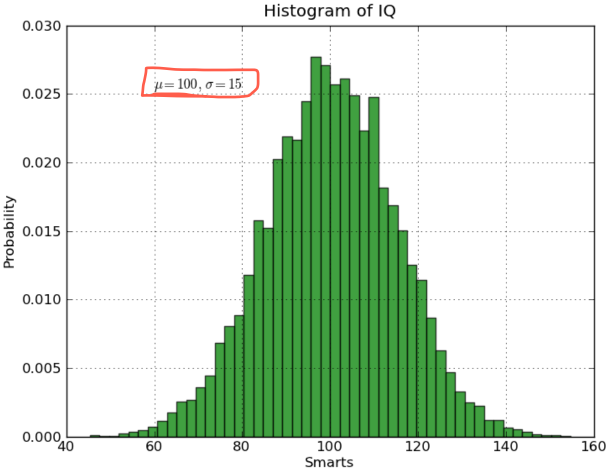
Lines are given with color and type

```
blue, green, red, cyan, magenta,
yellow, black, white

-   solid          --  dashed solid line
-.  dash-dot       :   dot line
o   circle         v   triangle
```

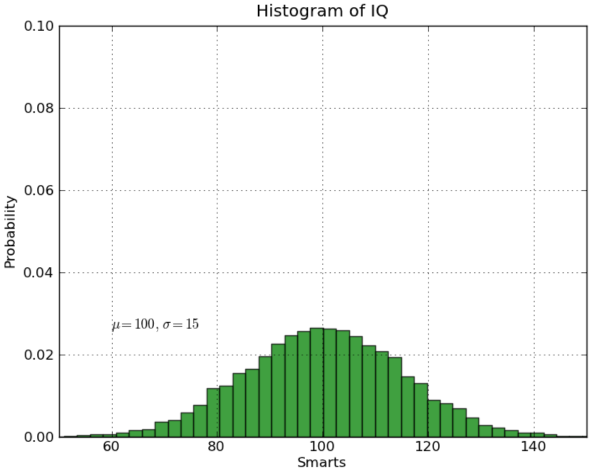
Graph

```
# fn: hist.py
import numpy as np
import pylab as plt

mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

n, bins, patches = plt.hist(x, 50, normed=1, facecolor='g', alpha=0.75);
t1 = np.arange( 0, 5, 0.1)
t2 = np.arange( 0, 5, 0.02)

plt.xlabel('Smarts'); plt.ylabel('Probability');
plt.title('Histogram of IQ');
plt.text(60, .025, r'$\mu=100,\ \sigma=15$');
plt.axis([40, 160, 0, 0.03]);
plt.grid(True); plt.show()
```

Graph


Histogram of IQ
$\mu=100,\ \sigma=15$
Probability
Smarts
0.030
0.025
0.020
0.015
0.010
0.005
0.000
40
60
80
100
120
140
160

Graph


Histogram of IQ
0.10
0.08
0.06
0.04
0.02
0.00
Probability
$\mu=100,\ \sigma=15$
60
80
100
120
140
Smarts

Graph

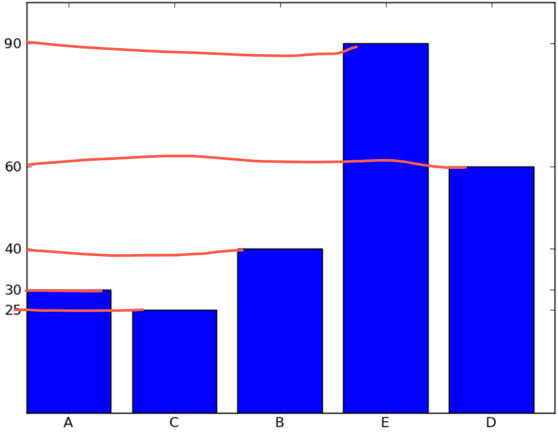
```
# fn: hist2.py
import numpy as np
import pylab as plt

mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)
n, bins, patches = plt.hist(x, 50, normed=1, facecolor='g', alpha=0.75);
t1 = np.arange( 0, 5, 0.1)
t2 = np.arange( 0, 5, 0.02)

plt.xlabel('Smarts'); plt.ylabel('Probability');
plt.title('Histogram of IQ');
plt.text(60, .025, r'$\mu=100,\ \sigma=15$');
plt.axis([40, 160, 0, 0.03]);
plt.ylim(0, 0.1); plt.xlim(50, 150)
plt.grid(True); plt.show()
```

ylim(0,1)

Graph

```
# fn: bar.py
import matplotlib.pyplot as plt
import numpy as np

dict = {'A':30, 'B':40, 'C':25, 'D':60, 'E':90};
for i, key in enumerate(dict):
    plt.bar(i, dict[key]);
plt.ylim([0, 100]);
plt.xticks(np.arange(len(dict))+0.4, dict.keys());
plt.yticks(dict.values());
plt.show()
```

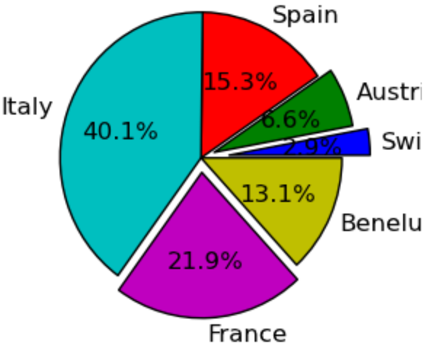
Graph



Graph

```
# fn: pie.py
import matplotlib.pyplot as plt
import numpy as np

plt.figure(figsize=(3,3));
x = [4, 9, 21, 55, 30, 18];
labels = ['Swiss', 'Austria', 'Spain', 'Italy', 'France', 'Benelux'];
# the offset fraction of the wedge from the center
explode = [0.2, .1, 0, 0, .1, 0];
plt.pie(x, labels=labels, explode=explode, autopct='%1.1f%%');
plt.show()
```

Graph



Summary

- Program statements in Python can be grouped into blocks using indentation.
- the fundamental ways to control a program's behavior, loop and conditional statements
- Control structures like loops and conditionals can be nested inside one another to any desired depth.
- Python and other languages keep track of nested function calls using a call stack.
- Programs can use `input` to get input from users interactively.
- Visualizing various graphs

Problem 1

Homework

Write a program that takes a positive integer N as input and draws N random integers in the interval $[1, 6]$ (both ends inclusive). N increase by 10 to 100. Answer the following questions.

- Count the frequency of each number and compute the fractions
- Plot both the frequency and the fraction of each number
- Discuss the changing trend as N increases

Use `random.randint(1,6)` from module `random` or `numpy.random.randint(1,7)` from module `numpy`.

Problem 2

Homework

Consider some game where each participant draws a series of random integers evenly distributed from 0 to 10, with the aim of getting the sum as close as possible to 21, but *not larger than 21*. You are out of the game if the sum passes 21. After each draw, you are told the number and is asked whether you want another draw or not. The one coming closest to 21 is the winner. Implement this game.

Use `random.randint(0,10)` from module `random` or `numpy.random.randint(0,11)` from module `numpy`.

Problem 3

Homework

There are some data D which are collected from a device.

$$D = \{(0, 0.5), (1, 2.0), (2, 1.0), (3, 1.5), (4, 7.5)\}$$

Your task is to write a program that fits a straight line to those data.

- Given a and b, make a function `compute_error(a,b,y)` that computes the error between the straight line $\boxed{f(x) = ax + b}$ and D.

$$e = \sum_{i=1}^{5} (ax_i + b - y_i)$$

- Plot a straight line $f(x)$ given a and b.
- Search for a and b such that e is minimized.