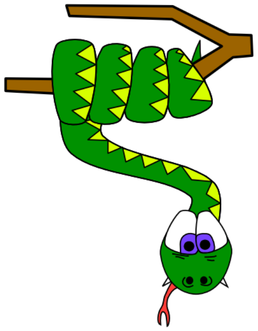
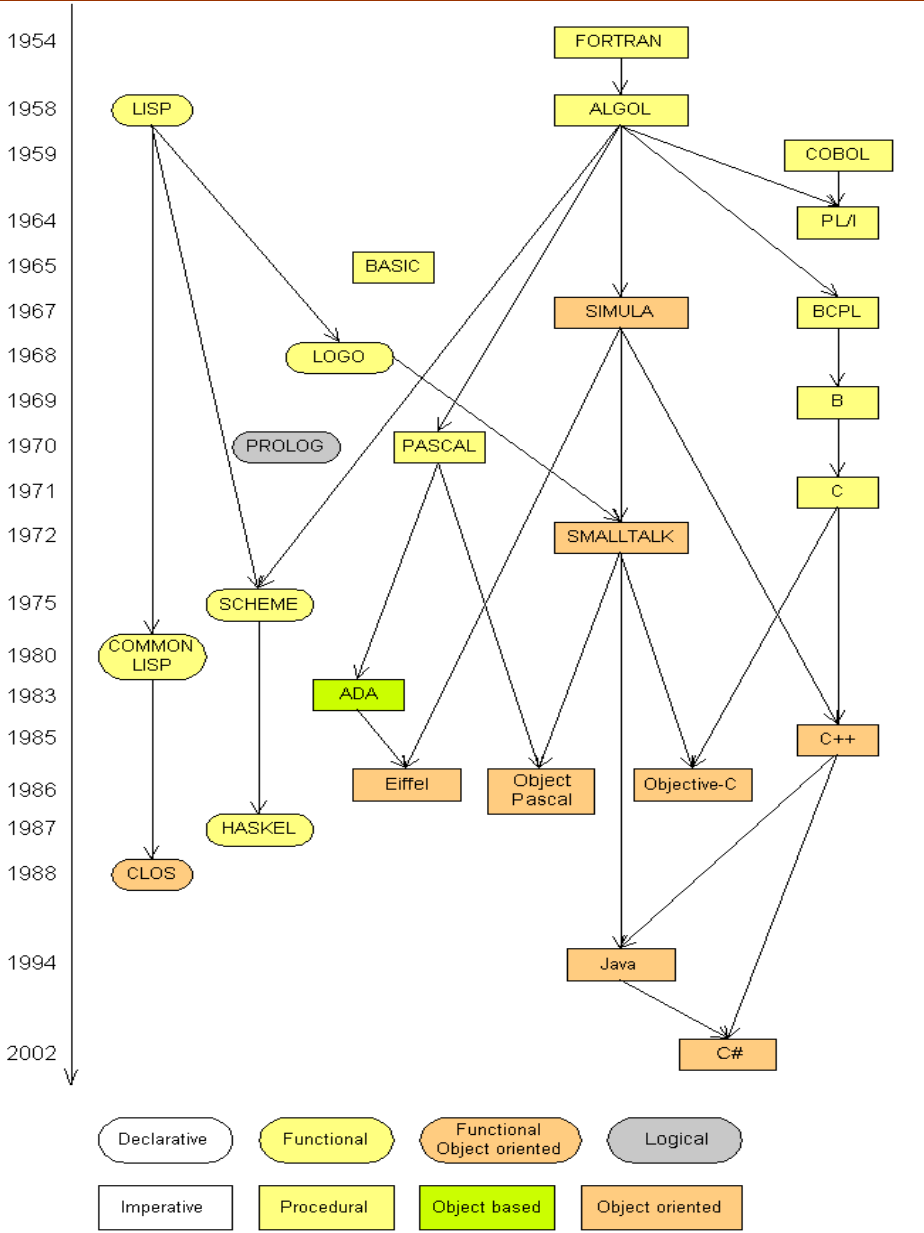


Introduction to Python Programming

Prof. Prakash Patel
Information Technology Department
Gandhinagar Institute of Technology

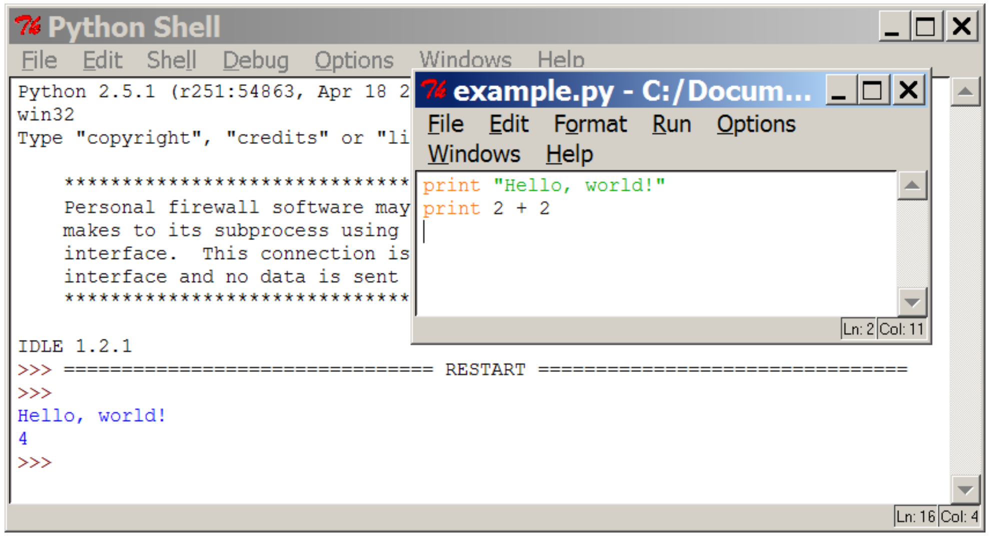
Languages

- Some influential ones:
  - FORTRAN
    - science / engineering
  - COBOL
    - business data
  - LISP
    - logic and AI
  - BASIC
    - a simple language



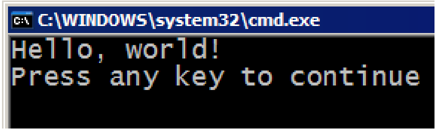
Programming basics

- **code** or **source code**: The sequence of instructions in a program.
- **syntax**: The set of legal structures and commands that can be used in a particular programming language.
- **output**: The messages printed to the user by a program.
- **console**: The text box onto which output is printed.
  - Some source code editors pop up the console as an external window, and others contain their own console window.



```
print "Hello, world!"
print 2 + 2
```

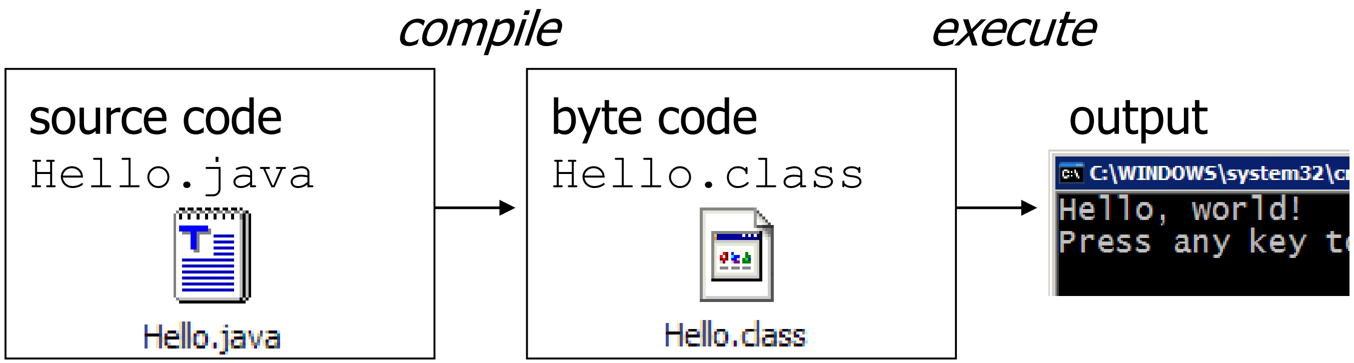
```
Python 2.5.1 (r251:54863, Apr 18 2
win32
Type "copyright", "credits" or "li

    ****************************
    Personal firewall software may
    makes to its subprocess using
    interface.  This connection is
    interface and no data is sent
    ****************************

IDLE 1.2.1
>>> ================================ RESTART ================================
>>>
Hello, world!
4
>>>
```



```
Hello, world!
Press any key to continue
```

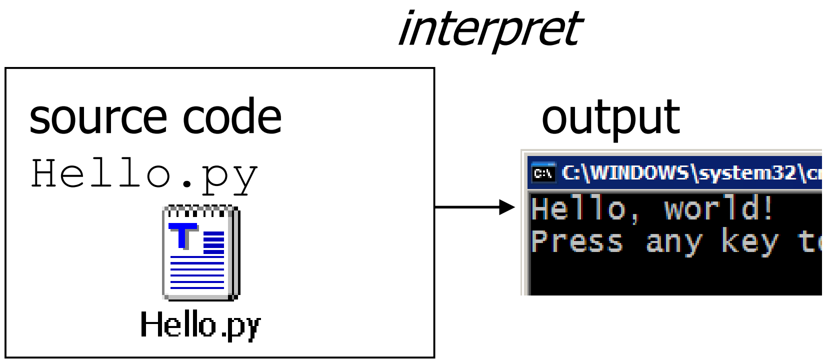
Compiling and interpreting

- Many languages require you to *compile* (translate) your program into a form that the machine understands.



```
Hello, world!
Press any key t
```

- Python is instead directly *interpreted* into machine instructions.



```
Hello, world!
Press any key t
```

What is Python?

- Python is a high-level, interpreted, interactive and object-oriented scripting language.
- Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.
- Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

History of Python

- Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, Unix shell, and other scripting languages.
- Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).
- Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

- **Python is Interpreted:** Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive:** You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented:** Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language:** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Python Features

- **Easy-to-learn:** Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read:** Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain:** Python's source code is fairly easy-to-maintain.
- **A broad standard library:** Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode:** Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

Python Features

- **Extendable:** You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases:** Python provides interfaces to all major commercial databases.
- **GUI Programming:** Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable:** Python provides a better structure and support for large programs than shell scripting.

Python Features

- Apart from the above-mentioned features, Python has a big list of good features, few are listed below:
- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Where Python is used?

- Testing microchips
- To powering Instagram
- To building video games
- Network Management
- Scientific Application
- As Scripting language
- Web Applications
- Network Servers
- Media tools
- Bittorrent written in Python

Getting Python

- The most up-to-date and current source code, binaries, documentation, news, etc.,is available on the official website of Python: http://www.python.org/.
- You can download Python documentation from www.python.org/doc/. The documentation is available in HTML, PDF, and PostScript formats.

Running Python

- There are different ways to start Python:

(1) Interactive Interpreter

- You can start Python from Unix, DOS, or any other system that provides you a command-line interpreter or shell window.
- Enter **python** the command line.
- Start coding right away in the interactive interpreter.

  - $python # Unix/Linux

    or

  - python% # Unix/Linux

    or

  - C:>python # Windows/DOS

(2) Script from the Command-line

- A Python script can be executed at command line by invoking the interpreter on your application, as in the following:

- $python script.py # Unix/Linuxor
- python% script.py # Unix/Linuxor
- C:>python script.py #Windows/DOS

First Python Program

- Let us execute programs in different modes of programming.
- **Interactive Mode Programming:**
- Invoking the interpreter without passing a script file as a parameter brings up the following prompt:
- $ python
- >>> print "Hello, Python!";
- If you are running new version of Python, then you need to use print statement with parenthesis as in **print ("Hello, Python!");**. However in Python version 2.4.3, this produces the following result:
- Hello, Python!

- **Script Mode Programming:**
- Let us write a simple Python program in a script. Python files have extension **.py**.
- Type the following source code in a test.py file:

```
print "Hello, Python!";
```

- Now, try to run this program as follows:
- $ python test.py
- This produces the following result:
- Hello, Python!

Multi-Line Statements

- Statements in Python typically end with a new line. Python does, however, allow the
- use of the line continuation character (\) to denote that the line should continue. For example:

-
  ```
  total = item_one + \
          item_two + \
          item_three
  ```
- Statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example:

-
  ```
  days = ['Monday', 'Tuesday', 'Wednesday',
          'Thursday', 'Friday']
  ```

Quotation in Python

- Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.
- The triple quotes are used to span the string across multiple lines. For example, all the following are legal:
- word = 'word'

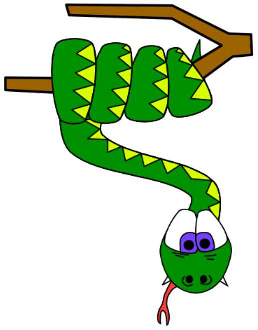
```
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences.""“
```

Comments in Python

- A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.
- #!/usr/bin/python
- # First comment
- print "Hello, Python!"; # second comment
- You can type a comment on the same line after a statement or expression:
- name = "Madisetti" # This is again comment

continue

- You can comment multiple lines as follows:
- # This is a comment.
- # This is a comment, too.
- # This is a comment, too.
- # I said that already.



Repetition (loops) and Selection (if/else)

The `for` loop

- **`for` loop**: Repeats a set of statements over a group of values.
  - Syntax:

    for ***variableName*** in ***groupOfValues***:
        statements

    - We indent the statements to be repeated with tabs or spaces.
    - ***variableName*** gives a name to each value, so you can refer to it in the ***statements***.
    - ***groupOfValues*** can be a range of integers, specified with the `range` function.
  - Example:

```
for x in range(1, 6):
    print x, "squared is", x * x
```

Output:

```
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
```

`range`

- The `range` function specifies a range of integers:
  - `range(`***start***`, `***stop***`)` - the integers between ***start*** (inclusive) and ***stop*** (exclusive)

- It can also accept a third value specifying the change between values.
  - `range(`***start***`, `***stop***`, `***step***`)` - the integers between ***start*** (inclusive) and ***stop*** (exclusive) by ***step***

- Example:

```
for x in range(5, 0, -1):
    print x
print "Blastoff!"
```

Output:

```
5
4
3
2
1
Blastoff!
```

Cumulative loops

- Some loops incrementally compute a value that is initialized outside the loop. This is sometimes called a *cumulative sum*.

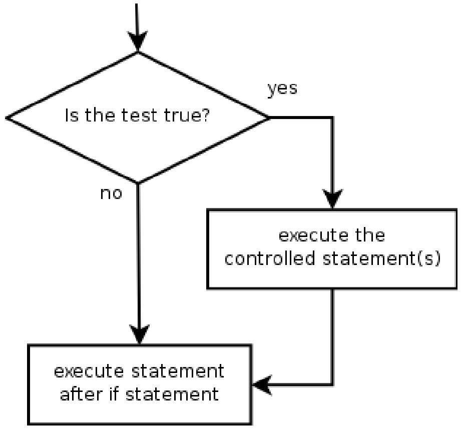
```
sum = 0
for i in range(1, 11):
    sum = sum + (i * i)
print "sum of first 10 squares is", sum

Output:
sum of first 10 squares is 385
```

- **Exercise:** Write a Python program that computes the factorial of an integer.

`if`

- **`if` statement**: Executes a group of statements only if a certain condition is true. Otherwise, the statements are skipped.
  - Syntax:

```
if condition:
    statements
```


Is the test true?
yes
no
execute the controlled statement(s)
execute statement after if statement

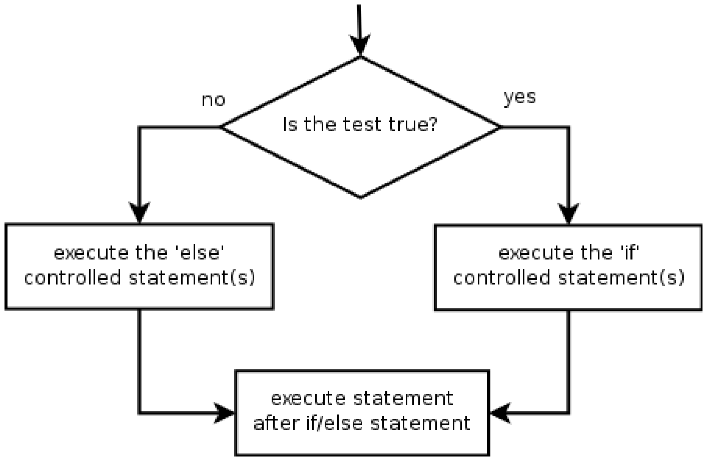
- Example:

```
gpa = 3.4
if gpa > 2.0:
    print "Your application is accepted."
```

if/else

- **`if/else` statement**: Executes one block of statements if a certain condition is True, and a second block of statements if it is False.
  - Syntax:

```
if condition:
    statements
else:
    statements
```



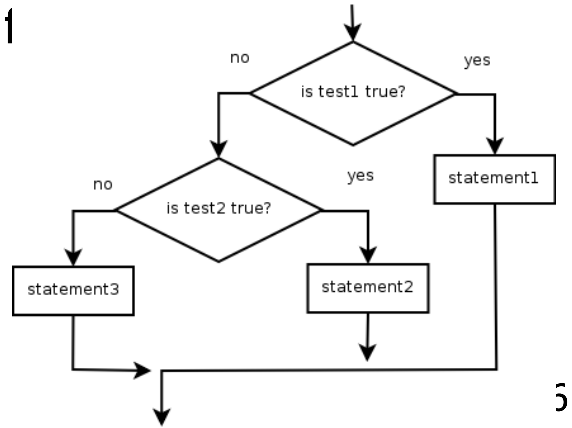
Is the test true?
no
yes
execute the 'else' controlled statement(s)
execute the 'if' controlled statement(s)
execute statement after if/else statement

- Example:

```
gpa = 1.4
if gpa > 2.0:
    print "Welcome to Mars University!"
else:
    print "Your application is denied."
```

- Multiple conditions can be chained with `elif` ("else if

```
if condition:
    statements
elif condition:
    statements
else:
    statements
```



while

- `while` **loop**: Executes a group of statements as long as a condition is True.
  - good for *indefinite loops* (repeat an unknown number of times)

- Syntax:

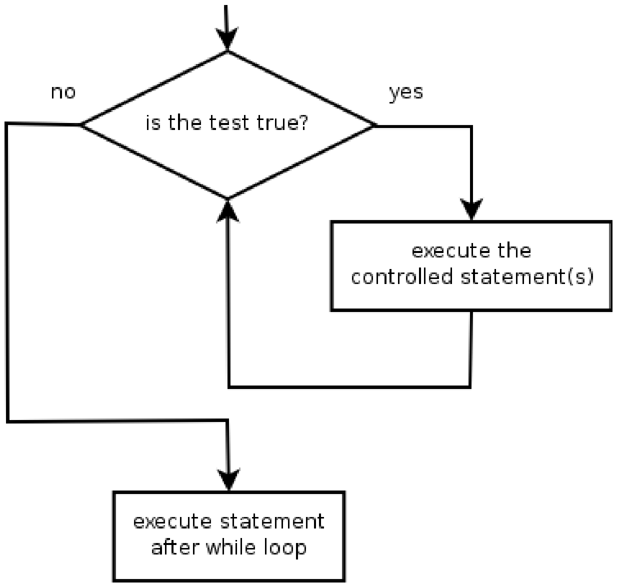
```
while condition:
    statements
```

- Example:

```
number = 1
while number < 200:
    print number,
    number = number * 2
```

  - Output:

```
1 2 4 8 16 32 64 128
```


no
is the test true?
yes
execute the controlled statement(s)
execute statement after while loop

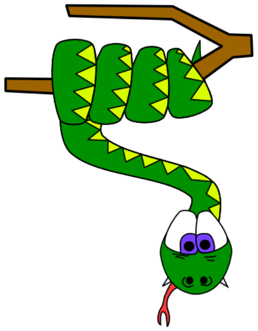
Logic

- Many logical expressions use *relational operators*:

Operator	Meaning	Example	Result
`==`	equals	`1 + 1 == 2`	`True`
`!=`	does not equal	`3.2 != 2.5`	`True`
`<`	less than	`10 < 5`	`False`
`>`	greater than	`10 > 5`	`True`
`<=`	less than or equal to	`126 <= 100`	`False`
`>=`	greater than or equal to	`5.0 >= 5.0`	`True`

- Logical expressions can be combined with *logical operators*:

Operator	Example	Result
`and`	`9 != 6 and 2 < 3`	`True`
`or`	`2 == 3 or -1 < 5`	`True`
`not`	`not 7 > 0`	`False`



Text and File Processing

Strings

- **string**: A sequence of text characters in a program.
  - Strings start and end with quotation mark `"` or apostrophe `'` characters.
  - Examples:

    ```
    "hello"
    "This is a string"
    "This, too, is a string.   It can be very long!"
    ```

- A string may not span across multiple lines or contain a " character.

  ```
  "This is not
  a legal String."
  ```

  ```
  "This is not a "legal" String either."
  ```

- A string can represent characters by preceding them with a backslash.
  - `\t` tab character
  - `\n` new line character
  - `\"` quotation mark character
  - `\\` backslash character

  - Example: `"Hello\tthere\nHow are you?"`

Indexes

- Characters in a string are numbered with *indexes* starting at 0:
  - Example:

```
name = "P. Diddy"
```

index	0	1	2	3	4	5	6	7
character	P	.		D	i	d	d	y

- Accessing an individual character of a string:

  variableName [***index***]

  - Example:

```
print name, "starts with", name[0]
```

Output:

```
P. Diddy starts with P
```

String properties

- `len(`***string***`)` - number of characters in a string (including spaces)
- `str.lower(`***string***`)` - lowercase version of a string
- `str.upper(`***string***`)` - uppercase version of a string

- Example:

```
name = "Martin Douglas Stepp"
length = len(name)
big_name = str.upper(name)
print big_name, "has", length, "characters"
```

Output:

```
MARTIN DOUGLAS STEPP has 20 characters
```

raw_input

- `raw_input` : Reads a string of text from user input.
  - Example:

```
name = raw_input("Howdy, pardner. What's yer name? ")
print name, "... what a silly name!"
```

    Output:

```
Howdy, pardner. What's yer name? Paris Hilton
Paris Hilton ... what a silly name!
```

Text processing

- **text processing**: Examining, editing, formatting text.
  - often uses loops that examine the characters of a string one by one

- A `for` loop can examine each character in a string in sequence.
  - Example:

```
for c in "booyah":
    print c
```

Output:

```
b
o
o
y
a
h
```

Strings and numbers

- `ord(`***text***`)` - converts a string into a number.
  - Example: `ord("a")` is `97`, `ord("b")` is `98`, ...
  - Characters map to numbers using standardized mappings such as *ASCII* and *Unicode*.
- `chr(`***number***`)` - converts a number into a string.
  - Example: `chr(99)` is `"c"`
- **Exercise:** Write a program that performs a rotation cypher.
  - e.g. `"Attack"` when rotated by 1 becomes `"buubdl"`

File processing

- Many programs handle data, which often comes from files.

- Reading the entire contents of a file:

  variableName `= open("`***filename***`").read()`

  Example:

```
file_text = open("bankaccount.txt").read()
```

Line-by-line processing

- Reading a file line-by-line:

```
for line in open("filename").readlines():
    statements
```

  Example:

```
count = 0
for line in open("bankaccount.txt").readlines():
    count = count + 1
print "The file contains", count, "lines."
```

- **Exercise:** Write a program to process a file of DNA text, such as:

```
ATGCAATTGCTCGATTAG
```

  - Count the percent of C+G present in the DNA.