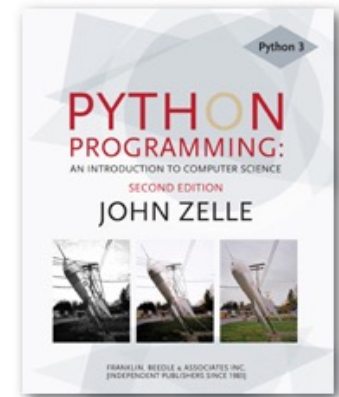


Python Programming: An Introduction to Computer Science



Chapter 1 Computers and Programs



Hardware Basics

- Input devices
 - Information is passed to the computer through keyboards, mice, etc.
- Output devices
 - Processed information is presented to the user through the monitor, printer, etc.



Hardware Basics

- *Fetch-Execute Cycle*
 - First instruction retrieved from memory
 - Decode the instruction to see what it represents
 - Appropriate action carried out.
 - Next instruction fetched, decoded, and executed.
 - Lather, rinse, repeat!



Programming Languages

- Natural language has ambiguity and imprecision problems when used to describe complex algorithms.
 - Programs expressed in an unambiguous, precise way using *programming languages*.
 - Every structure in programming language has a precise form, called its *syntax*
 - Every structure in programming language has a precise meaning, called its *semantics*.



Programming Languages

- Programming language like a code for writing the instructions the computer will follow.
 - Programmers will often refer to their program as *computer code*.
 - Process of writing an algorithm in a programming language often called *coding*.



Programming Languages

- *High-level* computer languages
 - Designed to be used and understood by humans
- Low-level language
 - Computer hardware can only understand a very low level language known as *machine language*

1.5 Programming languages

A list of programming languages (not full):

C++, Java, Python, Perl, Scheme, Visual Basic, Pascal, ...

- *high-level computer languages* i.e. close to our natural language

Computer hardware can only understand a very low-level *machine language*.

The *machine language* of a computer is created by the designers of the particular CPU. Each kind of computer has its own machine language.

1.5 Programming languages: Compilers and Interpreters

To add two numbers in Python: $c = a + b$

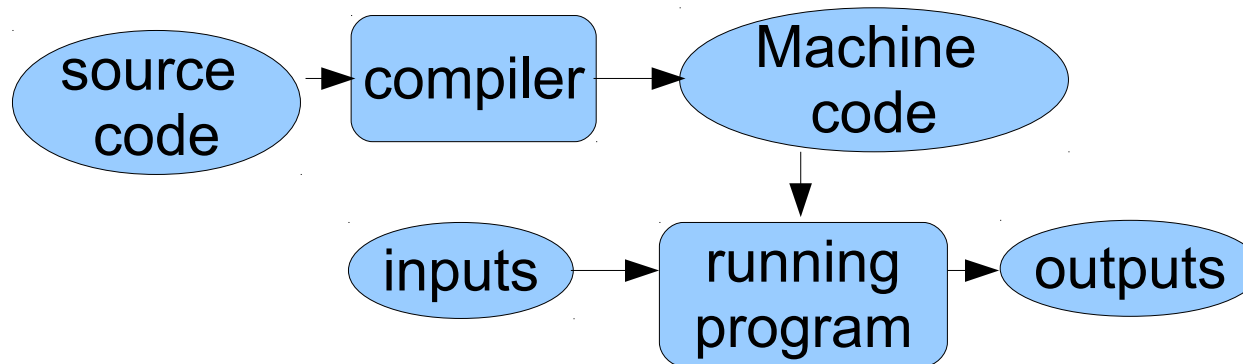
To add two numbers in machine language: (something like)
load the number from memory location 2001 into the CPU
load the number from memory location 2002 into the CPU
add the two numbers in the CPU
store the result into location 2003
- all these instructions and numbers are represented in *binary notation* as sequences of 0's and 1's.

Thus we need some way to *translate* high-level language into low-level language that the computer can execute. There are two ways to do it: *compile* or *interpret*

1.5 Programming languages: Compilers and Interpreters

Compiler

is a computer program that takes a program written in a high-level language and *translates* it into an equivalent program in the machine language *of some computer*.

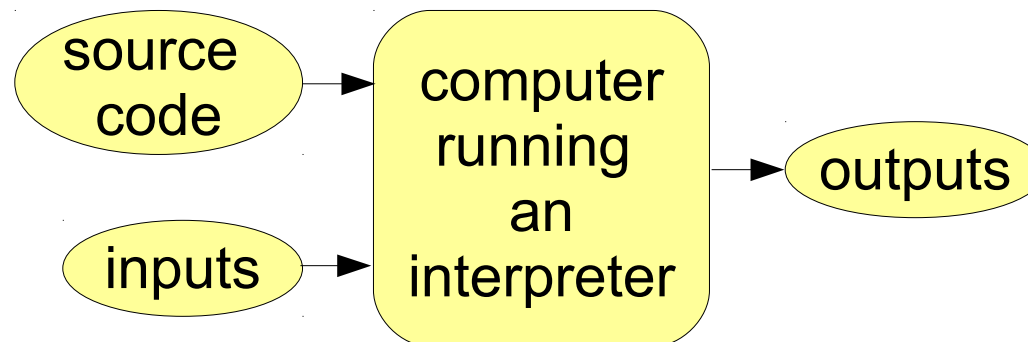


1.5 Programming languages: Compilers and Interpreters

Interpreter

is a computer program that simulates a computer that understands a high-level language.

Rather than translating the source program into a machine language equivalent, the interpreter analyses and executes source code instruction by instruction as necessary.





Programming Languages

- **Compiling vs. Interpreting**
 - Once program is compiled, it can be executed over and over without the source code or compiler. If it is interpreted, the source code and interpreter are needed each time the program runs
 - Compiled programs generally run faster since the translation of the source code happens only once.



Programming Languages

- Interpreted languages are part of a more flexible programming environment since they can be developed and run interactively
- Interpreted programs are more *portable*, meaning the executable code produced from a compiler for a Pentium won't run on a Mac, without recompiling. If a suitable interpreter already exists, the interpreted code can be run with no modifications.



The Magic of Python

When you start Python, you will see something like:

```
Python 3.1.2 (r312:79149, Mar 21 2010, 00:41:52) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```



The Magic of Python

- The “>>>” is a Python *prompt* indicating that Python is ready for us to give it a command. These commands are called *statements*.
- ```
>>> print("Hello, world")
Hello, world
>>> print(2+3)
5
>>> print("2+3=", 2+3)
2+3= 5
>>>
```



# The Magic of Python

---

- Usually we want to execute several statements together that solve a common problem. One way to do this is to use a *function*.
- ```
>>> def hello():  
    print("Hello")  
    print("Computers are Fun")
```

```
>>>
```



The Magic of Python

- ```
>>> def hello():
 print("Hello")
 print("Computers are Fun")
```

```
>>>
```

- The first line tells Python we are *defining* a new function called hello.
- The following lines are indented to show that they are part of the hello function.
- The blank line (hit enter twice) lets Python know the definition is finished.





# The Magic of Python

---

- ```
>>> def hello():  
    print("Hello")  
    print("Computers are Fun")
```

```
>>>
```

- Notice that nothing has happened yet! We've defined the function, but we haven't told Python to perform the function!

- A function is *invoked* by typing its name.

- ```
>>> hello()
Hello
Computers are Fun
>>>
```



# The Magic of Python

---

- What's the deal with the ()'s?
- Commands can have changeable parts called *parameters* that are placed between the ()'s.
- ```
>>> def greet(person):  
    print("Hello", person)  
    print ("How are you?")  
  
>>>
```



The Magic of Python

- ```
>>> greet("Terry")
Hello Terry
How are you?
>>> greet("Paula")
Hello Paula
How are you?
>>>
```
- **When we use parameters, we can customize the output of our function.**



# The Magic of Python

---

- When we exit the Python prompt, the functions we've defined cease to exist!
- Programs are usually composed of functions, *modules*, or *scripts* that are saved on disk so that they can be used again and again.
- A *module file* is a text file created in text editing software (saved as "plain text") that contains function definitions.
- A *programming environment* is designed to help programmers write programs and usually includes automatic indenting, highlighting, etc.



# The Magic of Python

---

```
File: chaos.py
A simple program illustrating chaotic behavior

def main():
 print("This program illustrates a chaotic function")
 x = eval(input("Enter a number between 0 and 1: "))
 for i in range(10):
 x = 3.9 * x * (1 - x)
 print(x)

main()
```

- We'll use *filename.py* when we save our work to indicate it's a Python program.
- In this code we're defining a new function called **main**.
- The `main()` at the end tells Python to run the code.



# The Magic of Python

---

```
>>>
```

```
This program illustrates a chaotic function
```

```
Enter a number between 0 and 1: .5
```

```
0.975
```

```
0.0950625
```

```
0.335499922266
```

```
0.869464925259
```

```
0.442633109113
```

```
0.962165255337
```

```
0.141972779362
```

```
0.4750843862
```

```
0.972578927537
```

```
0.104009713267
```

```
>>>
```



# Inside a Python Program

---

```
File: chaos.py
```

```
A simple program illustrating chaotic behavior
```

- Lines that start with `#` are called *comments*
- Intended for human readers and ignored by Python
- Python skips text from `#` to end of line



# Inside a Python Program

---

```
def main():
```

- Beginning of the definition of a function called *main*
- Since our program has only this one module, it could have been written without the *main* function.
- The use of *main* is customary, however.





# Inside a Python Program

---

```
print("This program illustrates a chaotic function")
```

- This line causes Python to print a message introducing the program.



# Inside a Python Program

---

```
x = eval(input("Enter a number between 0 and 1: "))
```

- **x** is an example of a *variable*
- A variable is used to assign a name to a value so that we can refer to it later.
- The quoted information is displayed, and the number typed in response is stored in **x**.



# Inside a Python Program

---

```
for i in range(10):
```

- For is a *loop* construct
- A loop tells Python to repeat the same thing over and over.
- In this example, the following code will be repeated 10 times.



# Inside a Python Program

---

```
x = 3.9 * x * (1 - x)
```

```
print(x)
```

- These lines are the *body* of the loop.
- The body of the loop is what gets repeated each time through the loop.
- The body of the loop is identified through indentation.
- The effect of the loop is the same as repeating this two lines 10 times!



# Inside a Python Program

---

```
for i in range(10):
 x = 3.9 * x * (1 - x)
 print(x)
```

- These are equivalent!

```
x = 3.9 * x * (1 - x)
print(x)
x = 3.9 * x * (1 - x)
print(x)
x = 3.9 * x * (1 - x)
print(x)
 x = 3.9 * x * (1 - x)
print(x)
x = 3.9 * x * (1 - x)
print(x)
x = 3.9 * x * (1 - x)
print(x)
x = 3.9 * x * (1 - x)
print(x)
x = 3.9 * x * (1 - x)
print(x)
x = 3.9 * x * (1 - x)
print(x)
x = 3.9 * x * (1 - x)
print(x)
```



# Inside a Python Program

---

```
x = 3.9 * x * (1 - x)
```

- This is called an *assignment* statement
- The part on the right-hand side (RHS) of the "=" is a mathematical expression.
- \* is used to indicate multiplication
- Once the value on the RHS is computed, it is stored back into (*assigned*) into x



# Inside a Python Program

---

```
main()
```

- This last line tells Python to *execute* the code in the function *main*



# Chaos and Computers

---

- The chaos.py program:

```
def main():
 print("This program illustrates a chaotic function")
 x = eval(input("Enter a number between 0 and 1: "))
 for i in range(10):
 x = 3.9 * x * (1 - x)
 print(x)
main()
```

- For any given input, returns 10 seemingly random numbers between 0 and 1
- It appears that the value of  $x$  is *chaotic*





# Chaos and Computers

---

- The function computed by program has the general form  $k(x)(1-x)$  where  $k$  is 3.9
- This type of function is known as a logistic function.
- Models certain kinds of unstable electronic circuits.
- Very small differences in initial value can have large differences in the output.



# Chaos and Computers

---

■ Input: 0.25

0.73125

0.76644140625

0.698135010439

0.82189581879

0.570894019197

0.955398748364

0.166186721954

0.540417912062

0.9686289303

0.118509010176

■ Input: 0.26

0.75036

0.73054749456

0.767706625733

0.6954993339

0.825942040734

0.560670965721

0.960644232282

0.147446875935

0.490254549376

0.974629602149