Streams and Pipes

Along the Stream by Sharon France

Streams

As a leaf is carried by a stream, whether the stream ends in a lake or in the sea, so too is the output of your program carried by a stream not knowing if the stream goes to the screen or to a file.

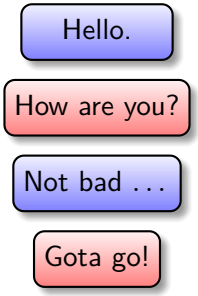
Washroom Wall (1995)
Quoted by Savitch

Input/Output

Read S&W Section 1.5 “Input and Output”

Input/Output

People communicate interactively and instinctively.



When people communicate with computers, it is is natural and valuable to mimic this. (Sometimes!) However, it is more complicated than it seems to program this interaction.

Complicated

1. Synchronizing the actors with respect to the arrow of time may be difficult.

2. The multiple systems involved are difficult to control and anticipate.
3. Specifying, or communicating precisely, the interactivity is difficult.

It is useful to break I/O into simpler pieces and learn simpler patterns of communication.

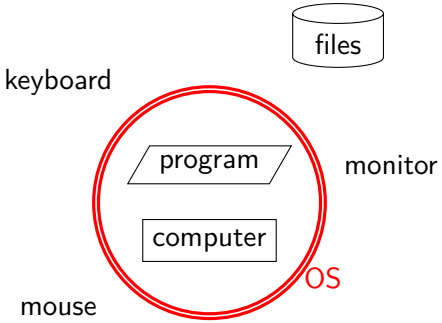
Streams

Programming I/O is kind of tedious and we might be tempted to give up and move on to more "pure" computational challenges. But programs need input and output.

- A program without input would always give the same result.
- A program without output would not be worth running as the user would get no answer.

Streams were invented with Unix operating system in the 1960s are universally used today as an organizing I/O principle.

Perhaps because people don't think much about I/O and are accustomed to complex GUIs, some minor points (like end-of-file) cause more trouble than they should. So we are careful here to explain the whole concept and hope to demonstrate why something so simple is actually quite profound.



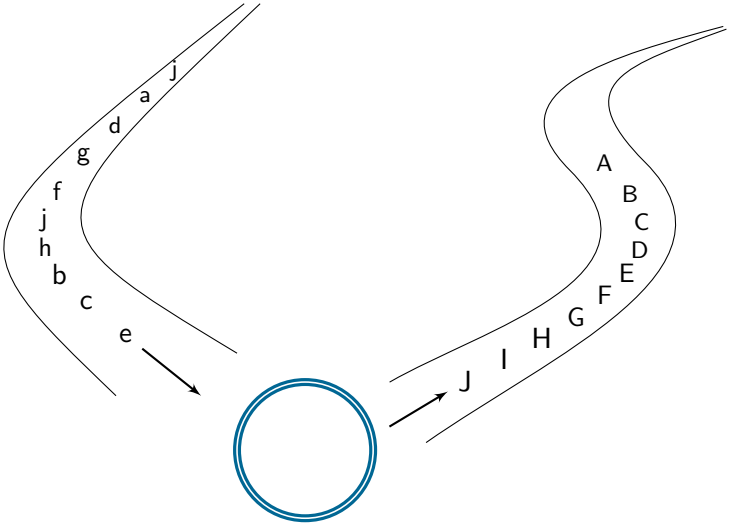
The program controls the computer, but needs the assistance of the operating system to communicate with the outside environment. A programming language is incapable of doing I/O except through the operating system.

Streams

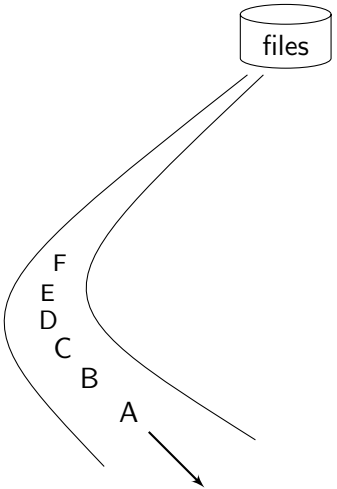
A stream is a convenient abstraction provided to a program by the operating system to make I/O easier and more uniform.

A *stream* is a conduit of data to or from a program. If the flow is into the program, the stream is called an *input stream*. If the flow is out of the program, the stream is called an *output stream*.

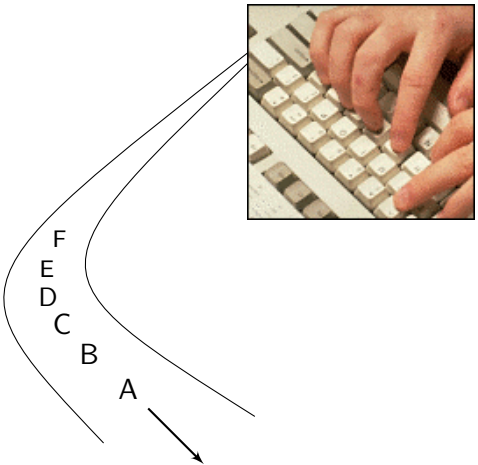
Input and Output Streams



Input Streams



Input Streams

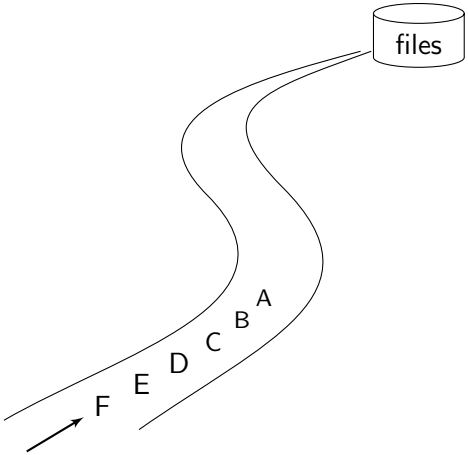


The data on an input stream can come from a person typing on the keyboard or from a file stored on the computer.

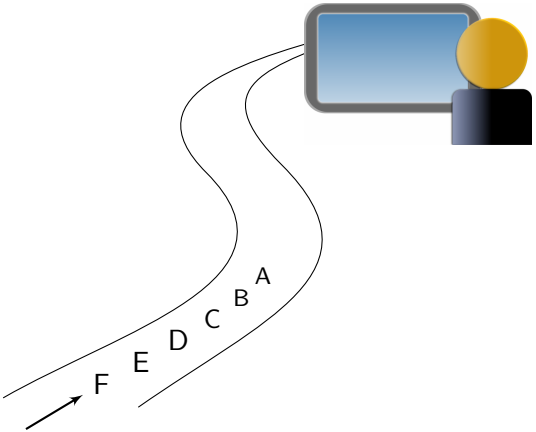
It is *more flexible* for the **user** to decide where the data comes from when the program is *executed*.

It is *less flexible* for the **programmer** to decide where the data comes from when the program is *written*.

Output Streams



Output Streams



The data on an output stream can go to a person at the monitor or to a file stored on the computer.

It is *more flexible* for the **user** to decide where the data goes to when the program is *executed*.

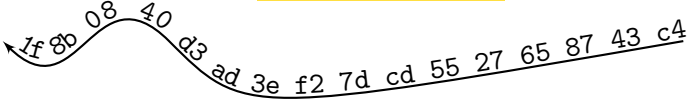
It is *less flexible* for the **programmer** to decide where the data goes to when the program is *written*.

Streams

Since the byte or octet (8-bits) is the smallest workable unit of (binary) data, the simplest view is that all data is a sequence of bytes. Files are a sequence of bytes; streams are a sequence of bytes.

These bytes make up the units of the common data types: characters, integers, etc.

```
1f 8b 08 40 d3
ad 3e f2 7d cd
55 27 65 87 43
c4
```



Medium/Ether

A medium is an agency or means of doing something. Writing is a medium of communication.

Un medio es una agencia de hacer algo. La escritura es un medio de comunicación.

Ein Medium ist eine Agentur oder ein Mittel, etwas zu tun. Schreiben ist ein Medium der Kommunikation.

Google Translate Chinese

Google Translate Arabic

In the process of communication, the medium is a channel or the means by which information (the message) is transmitted between a speaker or writer (the sender) and an audience (the receiver). In our case computation is expressed by the programmer to the computer by means of programming languages based on text.

Streams

A stream as realized in a programming language can hide many of the complicated facets of data in addition to the mere hardware. I/O details like

buffering, echoing, byte-order, character encoding, compression, and encryption

might easily be hidden from the programmer by a stream interface. The programmer may have access to these details via negociation with the OS, but each programming language differs in how these facilities are exposed, if at all.

Streams

Input and output is not always written and read by people. But for our purposes in this class it is convenient to focus on streams of text as opposed to other kinds of binary I/O data. Java makes I/O easy for text streams.

A *text stream* is a sequence of characters.

```
S t r e a m s ␣ a
r e ␣ s e q u e n
c e s
```

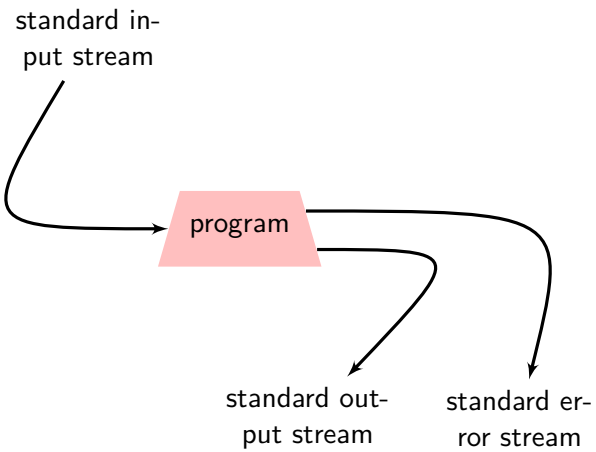
Do not assume that one byte encodes one character, although this is true for popular encodings like Latin-1, Mac-Roman, and others.

Standard I/O

A programming language these days normally assumes that the operating system provides three streams as a matter of course (and others streams upon request). This so-called *standard I/O* is divided into the *standard input stream* (associated with the keyboard by default), the *standard output stream* (associated with the display screen by default), and the *standard error stream* (also associated with the display screen by default).
One can completely ignore the existence of the standard error stream, if the program has no use for it.

Additional streams (both input and output) can be created by the programmer.

Standard I/O


standard input stream
program
standard output stream
standard error stream

Standard I/O in Languages

In any language there is a certain amount of mysterious boilerplate code required for taking advantage of the built-in standard I/O facilities. At first, one must find the right template and use it. All programs need I/O, even though the language mechanisms to enable the use of the I/O facilities are often less commonly used and may not be introduced to beginners.

```c
#include <stdio.h>

int main (void) {
   int c;
   for (;;) {
      c=fgetc(stdin);
      if (c==EOF) break;
   }
   fputc('A', stdout);
   fputc('X', stderr);
}
```

```
#include <iostream>

using std::cout;
using std::cin;
using std::cerr;

int main (void) {
   char c;
   while (cin.eof()) {
      cin>>c;
   }
   cout << 'A';
   cerr << 'X';
}
```

```
from sys import stdin, stdout, stderr
while True:
  ch = stdin.read(1)
  if not ch: break  # empty string = False
stdout.write ("A")
stderr.write ("X")
```

```
from sys import stdin, stdout

for line in stdin:
    stdout.write (line)  # write the line (with nl)
    tk1, tk2 = line.split()
    n, k, l = [int(i) for i in line.split()]
```

The One-And-One-Half Loop (Python)

My recommendation because it has no code duplication. Avoid `break`, but this has a single exit.

```
from sys import stdin, stdout

while True:
    line = stdin.readline()
    if (line == "sentinel"): break
    stdout.write (line)
```

```
from sys import stdin, stdout

line = stdin.readline()  # "Priming the Pump"
while line != "sentinel":
    stdout.write (line)
    line = stdin.readline()
```

The One-And-One-Half Loop

Alternatives in Python

```
from sys import stdin, stdout
from itertools import takewhile

# The 'takewhile' approach is hard to read and not "Pythonic"
# One liners are for "Code Golf" and not "real" code.
# [However, expressions are better than statements; see funct
for line in takewhile (lambda line: line[:-1]!="sentinel", st
   print (line)

from sys import stdin, stdout

# An unpleasant idiom found in C code.
# Walrus operator from PEP 527, 2018
while ((line:=stdin.readline()) != "sentinel\n"):
   print (line)
```

Streams in Java

The concept of a stream is realized in Java with two classes

java.io.InputStream java.io.OutputStream

The standard I/O streams are defined in the Java class `java.lang.System`, and they are:

```
InputStream in;
PrintStream out;
PrintStream err;
```

(A `PrintStream` is a special kind of `OutputStream`.)

Java does not treat the two kinds of streams equally.
The output streams are prepared in advance for simple text output. The input stream is not prepared in advance for simple text input and so the user usually has some preparation to do in order to facilitate the use of the standard input stream.

Inside a Java program the end-of-stream signal (aka end-of-file signal) from the operating system can be detected as follows:

```
// Create a reader for the standard input stream
final InputStreamReader reader =
   new InputStreamReader (System.in, "LATIN-1");

for (;;) {
   // read one character
   final int c = reader.read();
   // Exit loop on end-of-file
   if (c == -1) break;
}

System.out.print ('A');
System.err.print ('X');
```

Java uses a particular kind of input stream called “readers” especially for text.
Java uses Unicode internally for all characters and will translate from any character set, e.g, Latin-1, for you.
The Java method `read()` will return the integer -1 when the end of the input stream is reached.

For various reasons: convenience, efficiency, exception handling, etc; there is a better approach to text I/O that should used. It is illustrated in the next program.

Scanner Class

```
import java.util.Scanner;

public final class Scan {

   public static void main (final String[] args) {

      // System     is in the package java.lang
      // System.in  is a java.io.InputStream

      final Scanner stdin =
          new Scanner (System.in);
   }
}
```

This program interprets the standard input stream as text, i.e., of stream of characters. Java uses the Unicode character encoding internally, but what character encoding does it use to interpret the input stream? Greek characters, Japanese characters?

Scanner Class

Java interprets the characters in the default character set derived from the underlying operating system. So the same program on the same input may behave differently on different operating systems. Although unlikely, this is not the best programming practice.

We promise in this class that all input will be encoded in the US ASCII character set. (This is less likely to cause a problem if you are developing test cases on a platform with a different character set.)

In Java it is easy to make a program assume the input is encoded in US ASCII, and then the program will behave the same regardless of what platform it is executed on.

Scanner Class

To construct a scanner class which assumes the input is encoded n US ASCII, provide the name of the character encoding as the second argument.

```
import java.util.Scanner;

public final class Scan {

   public static void main (final String[] args) {

      // System     is in the package java.lang
      // System.in  is a java.io.InputStream

      final Scanner stdin =
          new Scanner (System.in, "US-ASCII");
   }
}
```

Scanner Class

```
import java.util.Scanner;
import static java.nio.charset.StandardCharsets.US_ASCII;

public final class Scan {

   public static void main (final String[] args) {

      // System      is in the package java.lang
      // System.in   is a java.io.InputStream

      final Scanner stdin =
          new Scanner (System.in, US_ASCII);
   }
}
```

Slightly better as we get static checking of the name of the character set. Though we wish to avoid `import static`.

Scanner Class

```
import java.util.Scanner;
import static java.nio.charset.StandardCharsets.US_ASCII;

public final class Scan {

   public static void main (final String[] args) {
      try (final Scanner stdin =
            new Scanner (System.in, US_ASCII)) {
         // Use the scanner here; when we reach the end
         // of the statement, the resources are returned.
      }
   }
}
```

It is good practice to release all resources (like the input stream) after you are done with it. But in this course, the programs will end soon anyway, so there is not much point to it.

```java
import java.util.Scanner;

public final class CopyText {

   public static void main (final String[] args) {

      // System.in  is a java.io.InputStream
      // System.out is a java.io.PrintStream

      final Scanner stdin =
          new Scanner(System.in, "US-ASCII");

      // Read standard input stream line by line
      while (stdin.hasNextLine()) {
         final String line = stdin.nextLine();
         System.out.println (line);
      }
   }
}
```

(This program does not buffer the standard input; for large input data, performance improvement can be obtained by buffering, i.e., using the Java class `BufferedInputStream`).

Avoid the method `String.split()` and the class `StringTokenizer`, when using the class `Scanner`, as `Scanner` does more and it is easier.

By default the scanner class breaks the input into tokens separated by white space.

Occasionally, one needs something else like entries separated by commas (and new lines).

```
import java.util.Scanner;

public class CommaDelimited {
   // Comma, or Unix line or DOS line terminator
   private final static String DELIM =
     ",|\\n|\\n\\r";
   public static void main (String[] args) {
      final Scanner stdin =
         new Scanner(System.in, "US-ASCII").
           useDelimiter(DELIM);

      // Read comma separated tokens in stdin
      while (stdin.hasNext()) {
         final String entry = stdin.next();
         System.out.println (entry);
      }
   }
}
```

Never mislead the reader or yourself with your choice of names for variables. For example, `keyboard` is a particularly bad choice to name a scanner associated with the standard input stream. The program has no control on where the input on the standard input stream comes from. Just calling the scanner `keyboard` does make the input come the keyboard.

```
final Scanner stdin =
   new Scanner(System.in,"US-ASCII").
      useDelimiter(DELIM);
```

NB. Chaining methods in this way is called *method cascading* in object-oriented languages. The method `useDelimiter()` returns `this` rather than be a less flexible `void` method.

File Redirection

You can control where the operating system gets the characters it puts in the input stream and where it puts the characters from the output stream. This is called *file redirection*. For example, on the Unix and Windows command line you can request that the standard input come from a file.

```
java ClassName < data
```

Now the characters found in the standard input come from the file named data. (The first version of Unix in 1970 already had file redirection for two I/O streams.)

File Redirection

The standard output is associated with the computer display by default, but can be associated with the file as in

```
java ClassName > output
```

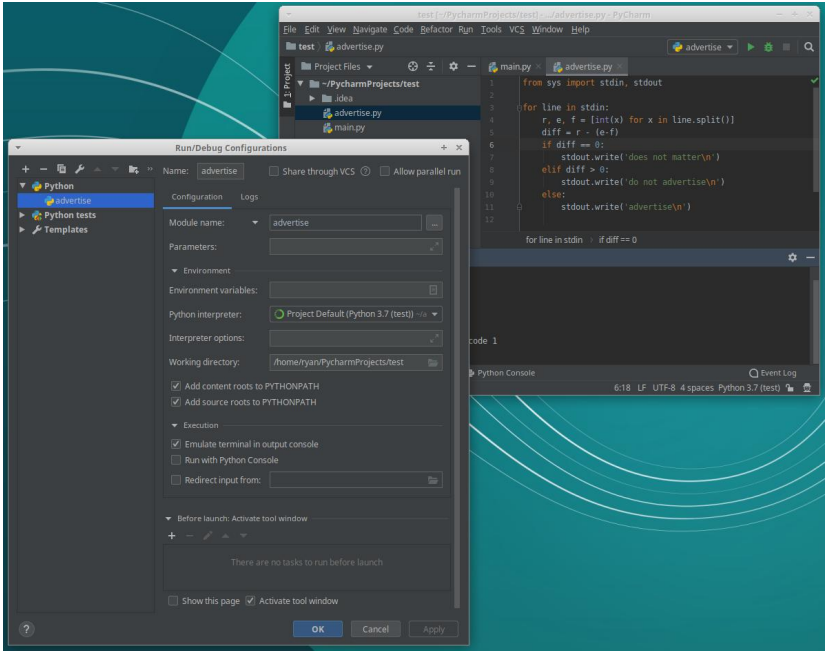
Now the output is collected in the file named `output`. Since keystrokes are echoed on the display, it is sometimes hard for the user to distinguish which characters on the display correspond to the standard input and which characters to the standard output. From the point of view of the program no such confusion exists.

File Redirection

Or, both.

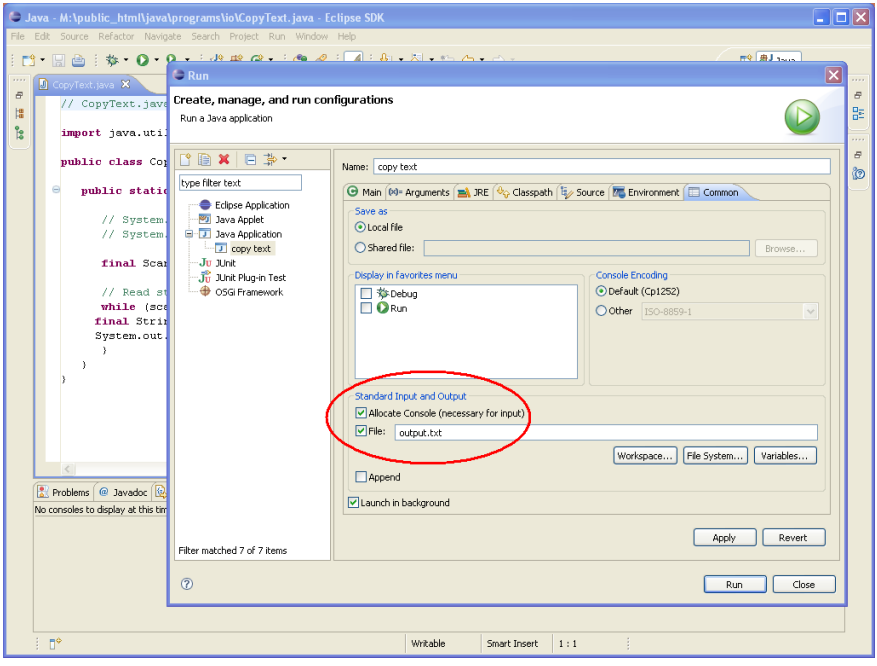
```
java ClassName < input > output
```

Test this way if you really want to see what is going on.


test [~/PycharmProjects/test] - .../advertise.py - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
test advertise.py
advertise
1: Project
Project Files
~/PycharmProjects/test
.idea
advertise.py
main.py
main.py
advertise.py
from sys import stdin, stdout
for line in stdin:
r, e, f = [int(x) for x in line.split()]
diff = r - (e-f)
if diff == 0:
stdout.write('does not matter\n')
elif diff > 0:
stdout.write('do not advertise\n')
else:
stdout.write('advertise\n')
for line in stdin
if diff == 0
code 1
Python Console
Event Log
6:18 LF UTF-8 4 spaces Python 3.7 (test)
Run/Debug Configurations
Name: advertise
Share through VCS
Allow parallel run
Python
advertise
Python tests
Templates
Configuration
Logs
Module name: advertise
Parameters:
Environment
Environment variables:
Python interpreter: Project Default (Python 3.7 (test))
Interpreter options:
Working directory: /home/ryan/PycharmProjects/test
Add content roots to PYTHONPATH
Add source roots to PYTHONPATH
Execution
Emulate terminal in output console
Run with Python Console
Redirect input from:
Before launch: Activate tool window
There are no tasks to run before launch
Show this page
Activate tool window
OK
Cancel
Apply

Eclipse

In additions to taking over the development of Java programs, Eclipse takes over the responsibility of the command shell in communicating to the operating system the command to execute, the arguments to the program, the environment of the process, the redirection of standard I/O.
Regrettable, Eclipse allows for the redirection of only the standard output, not the standard error nor the standard input. (Cutting and pasting in the console is a crude work-around.)


Java - M:\public_html\java\programs\io\CopyText.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
CopyText.java
// CopyText.java
import java.util
public class Co
public static
// System.
// System.
final Scan
// Read st
while (sc
final Strin
System.out.
}
}
Problems
Javadoc
No consoles to display at this tim
Run
Create, manage, and run configurations
Run a Java application
type filter text
Eclipse Application
Java Applet
Java Application
copy text
JUnit
JUnit Plug-in Test
OSGi Framework
Filter matched 7 of 7 items
Name: copy text
Main
Arguments
JRE
Classpath
Source
Environment
Common
Save as
Local file
Shared file:
Browse...
Display in favorites menu
Debug
Run
Console Encoding
Default (Cp1252)
Other
ISO-8859-1
Standard Input and Output
Allocate Console (necessary for input)
File:
output.txt
Workspace...
File System...
Variables...
Append
Launch in background
Apply
Revert
Run
Close
Writable
Smart Insert
1 : 1

Command Line/Shell

The user asks the operating system to run a program. There are numerous variations possible when a user runs a program. Often the operating system provides a command line interpreter (a program!) which the user can use to direct the execution by the operating system of applications on a computer.

Redirection in Bash

Your command line shell may have lots of features.

```
 <   filename   # redirect stdin from file
1>   filename   # redirect stdout to file
1>>  filename   # redirect and append stdout to file
2>   filename
2>>  filename
&>   filename   # redirect stdout and stderr to file
&>>  filename

2>&1            # redirect stderr into stdout
```

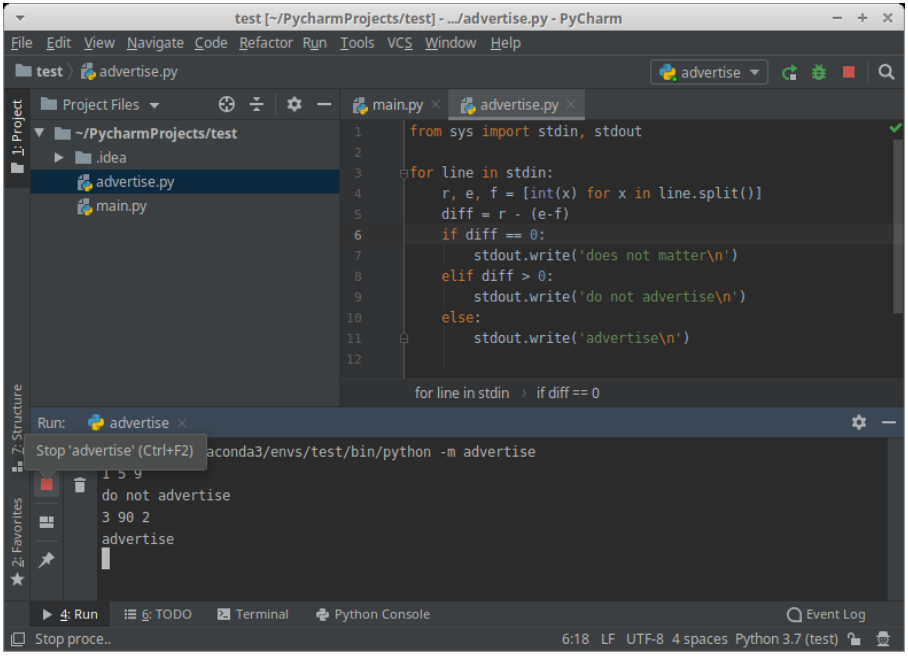
When the standard input comes from the keyboard a signal of some kind is needed to indicate from the interactive user when the end-of-file is reached because the length of the stream cannot be known in advance. This signal is understood by the operating system which then passes it to the program. Different operating systems provide different mechanisms for doing this. In Unix the usual signal is typing control-D (in Windows, control-Z) on a line by itself.

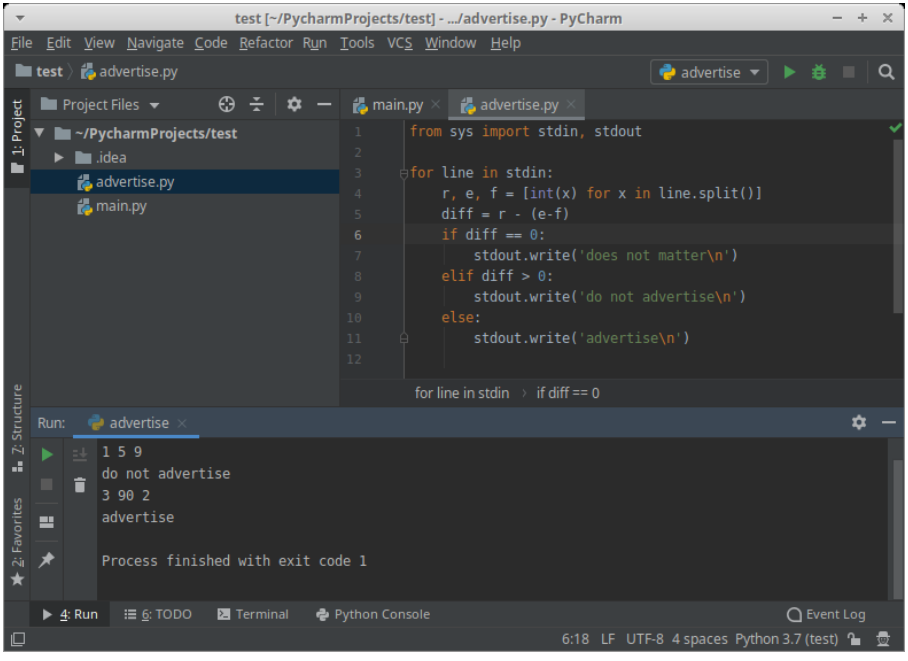
In Unix the user can change the keystrokes used to communicate with the operating system.

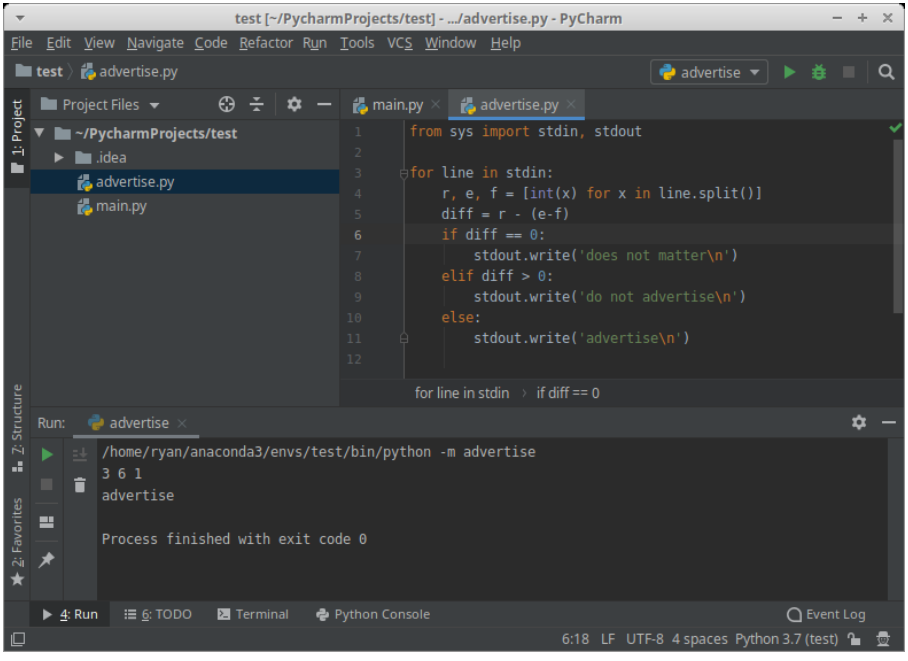
```
broadside> stty -a
speed 38400 baud; rows 38; columns 80;
intr = ^C; quit = ^\; erase = ^H; kill = ^U; eof = ^D; eol = <undef>;
eol2 = <undef>; start = ^Q; stop = ^S; susp = ^Z; dsusp = ^Y; rprnt = ^R;
werase = ^W; lnext = ^V; flush = ^O; status = ^T; min = 1; time = 0;
parenb -parodd cs8 hupcl -cstopb cread -clocal -crtscts
-ignbrk brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon -ixoff
-ixany -imaxbel
opost -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -tostop -echoprt echoctl
echoke
```

Inter-process communication in Unix is done through signals. And the shell (a process) can communicate to the terminal driver (a process) which can modify a data structure indicating the stream has run dry. In operating systems class one learns more about signals.

Thou shalt not kill
your program.


test [~/PycharmProjects/test] - .../advertise.py - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
test advertise.py
advertise
Project Files
1: Project
~/PycharmProjects/test
.idea
advertise.py
main.py
main.py
advertise.py
from sys import stdin, stdout
for line in stdin:
r, e, f = [int(x) for x in line.split()]
diff = r - (e-f)
if diff == 0:
stdout.write('does not matter\n')
elif diff > 0:
stdout.write('do not advertise\n')
else:
stdout.write('advertise\n')
for line in stdin
if diff == 0
Run:
advertise
Stop 'advertise' (Ctrl+F2)
aconda3/envs/test/bin/python -m advertise
1 5 9
do not advertise
3 90 2
advertise
7: Structure
2: Favorites
4: Run
6: TODO
Terminal
Python Console
Event Log
Stop proce..
6:18 LF UTF-8 4 spaces Python 3.7 (test)


test [~/PycharmProjects/test] - .../advertise.py - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
test advertise.py
advertise
Project Files
~/PycharmProjects/test
.idea
advertise.py
main.py
main.py
advertise.py
from sys import stdin, stdout
for line in stdin:
r, e, f = [int(x) for x in line.split()]
diff = r - (e-f)
if diff == 0:
stdout.write('does not matter\n')
elif diff > 0:
stdout.write('do not advertise\n')
else:
stdout.write('advertise\n')
for line in stdin
if diff == 0
Run:
advertise
1 5 9
do not advertise
3 90 2
advertise
Process finished with exit code 1
4: Run
6: TODO
Terminal
Python Console
Event Log
6:18 LF UTF-8 4 spaces Python 3.7 (test)
1: Project
7: Structure
2: Favorites


test [~/PycharmProjects/test] - .../advertise.py - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
test advertise.py
advertise
Project Files
~/PycharmProjects/test
.idea
advertise.py
main.py
main.py
advertise.py
from sys import stdin, stdout
for line in stdin:
r, e, f = [int(x) for x in line.split()]
diff = r - (e-f)
if diff == 0:
stdout.write('does not matter\n')
elif diff > 0:
stdout.write('do not advertise\n')
else:
stdout.write('advertise\n')
for line in stdin
if diff == 0
Run:
advertise
/home/ryan/anaconda3/envs/test/bin/python -m advertise
3 6 1
advertise
Process finished with exit code 0
4: Run
6: TODO
Terminal
Python Console
Event Log
6:18 LF UTF-8 4 spaces Python 3.7 (test)
1: Project
7: Structure
2: Favorites

Console

Looking at the console (display screen) can be confusing

```
maelstrom> java Confusing
Characters printed b
```

Three parallel processes may simultaneously be writing on the console.

Console

Looking at the console (display screen) can be confusing

```
maelstrom> java Confusing
Characters printed b Echoed in
```

Three parallel processes may simultaneously be writing on the console.

Console

Looking at the console (display screen) can be confusing

```
maelstrom> java Confusing
Characters printed b Echoed in y println() are seen
```

Three parallel processes may simultaneously be writing on the console.

Console

Looking at the console (display screen) can be confusing

```
maelstrom> java Confusing
Characters printed b Echoed in y println() are seen
put characters
```

Three parallel processes may simultaneously be writing on the console.

Console

Looking at the console (display screen) can be confusing

```
maelstrom> java Confusing
Characters printed b Echoed in y println() are seen
put characters Output sent to the
```

Three parallel processes may simultaneously be writing on the console.

Console

Looking at the console (display screen) can be confusing

```
maelstrom> java Confusing
Characters printed b Echoed in y println() are seen
put characters Output sent to the
 on the console.
```

Three parallel processes may simultaneously be writing on the console.

Console

Looking at the console (display screen) can be confusing

```
maelstrom> java Confusing
Characters printed b Echoed in y println() are seen
put characters Output sent to the
 on the console.  also appear
```

Three parallel processes may simultaneously be writing on the console.

Console

Looking at the console (display screen) can be confusing

```
maelstrom> java Confusing
Characters printed b Echoed in y println() are seen
put characters Output sent to the
 on the console.  also appear
on the console.
```

Three parallel processes may simultaneously be writing on the console.

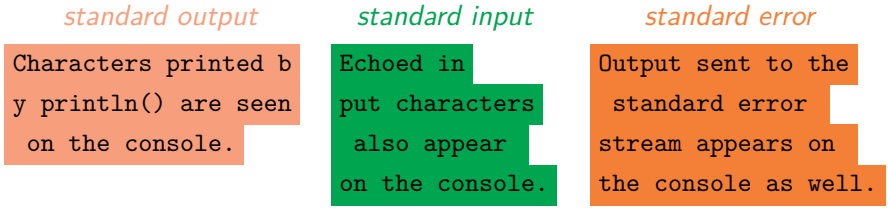
Console

Looking at the console (display screen) can be confusing

```
maelstrom> java Confusing
Characters printed b Echoed in y println() are seen
put characters Output sent to the
 on the console.  also appear
on the console.  standard error stream appears
 on the console as well.
```

Three parallel processes may simultaneously be writing on the console.

Programs that don't care what the user console looks like are simpler to write. It is easier to specify the contents of streams without reference to the other streams.



An interactive program may be hard to write and requires understanding exactly how the OS implements echoing, buffering, and flushing in order to achieve a specific temporal order.

For this reason I personally favor project descriptions that specify the contents of the input stream and specify the contents of the output stream independently. The programmer can choose to read and to write when it is most convenient or efficient. I avoid assigning interactive programs and programs that prompt for input because these programs tend to require tedious detail and specification of time-dependent behavior. However, sometimes such programs are more convenient for applications used by people. Many programs do not require an interface for humans.
Writing programs that are easy to use is important. This is a topic studied in more detail in the field of human-computer interaction and in classes on building graphical user-interfaces. Here we focus on basic programming and keep things simple.

Thou ſhalt not prompt the uſer.

Because this messes up the output stream. (There is no third output stream for messages to the user, unless that is considered an error.) Labeling the output is polite for human readers. But, spontaneous deviation from the requirements makes a program fail all test cases!

This is really a corollary of ...

Thou shalt care about every output character.

Creating Additional Streams

It is easy for the Java program to create input and output streams associated with files in the computer's file system or with a network connection to another program. (There is practically no use for creating additional streams associated with the keyboard or display device.)

The necessary classes from the Java package `java.io` and are:

```
FileReader (String file_name)
FileWriter (String file_name, boolean append)
```

```
String file_name1, file_name2;

try (
   final BufferedReader reader = new BufferedReader (
      new FileReader(file_name1));
   final PrintWriter    writer = new PrintWriter (
      new BufferedWriter (
         new FileWriter(file_name2, false)));
) {
   while (true) {
      final String line = reader.readLine();
      if (line==null) break;
      writer.println (line);
   }
}
```

```
try {
   String file_name1, file_name2;
   BufferedReader reader = new BufferedReader (
      new FileReader(file_name1));
   PrintWriter    writer = new PrintWriter (
      new BufferedWriter (
         new FileWriter(file_name2, false)));
   while (true) {
      final String line = reader.readLine();
      if (line==null) break;
      writer.println (line);
   }
   reader.close();
   writer.close();
} catch (IOException ex) {
   System.err.println(ex);
}
```

Thou shalt not kill
your program.

Thou shalt not prompt the user.

Also, no input validation is required (unless specified).

In Java (and in other modern languages):

- no buffer overflows, stack smashing;
- no dangling pointers;
- no information or memory disclosure;
- no crashes or core dumps.

Nonetheless, ...

Thou shalt program defensively.

Strong typing, the `assert` statement, and clear communication of preconditions are a good ideas.

Pipes

It is not possible to appreciate the value of the stream abstraction without understanding the notion of a *pipe*.

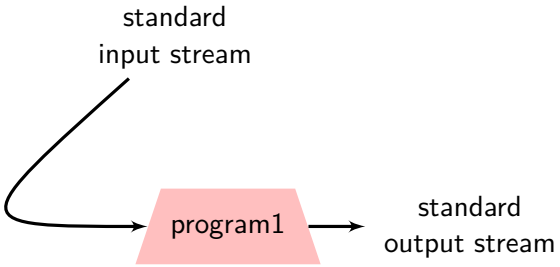
Pipes

> *One of the most widely admired contributions of Unix to the culture of operating systems and command languages is the pipe, as used in a pipeline of commands.*
>
> Dennis M. Ritchie

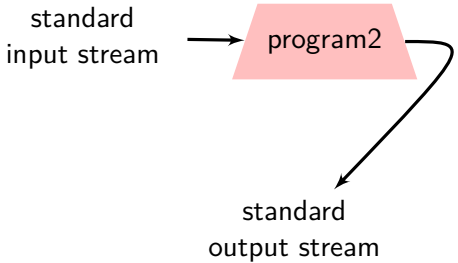
The operating system can connect the output stream of one program with the input stream of another program; these connections are called *pipes*. Small, well-designed programs can be fit together in many different ways to accomplish complex tasks.


DENNIS RITCHIE &
KEN THOMPSON
UNIX
Inventors of UNIX.
acm
A.M.
TURING
AWARD
1983
UNIX
BELL LABS
SENSITIVE

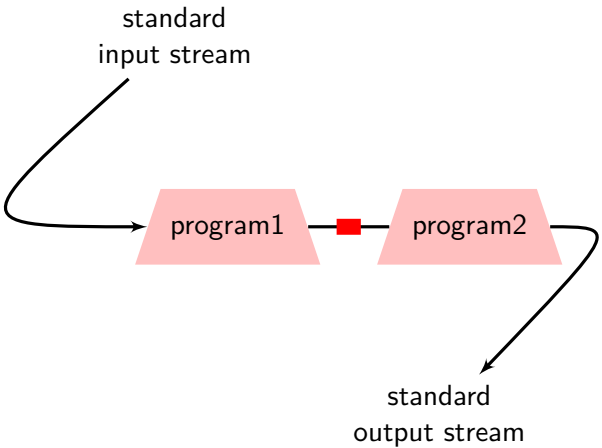
Pipes



Pipes



Pipes


standard
input stream
program1
program2
standard
output stream

System Design

Composing programming blocks to get new programming blocks is important to building large systems.

Don't write one monolithic program to solve one problem. Well-designed programs can work together to solve many problems without constantly writing and re-writing programs.

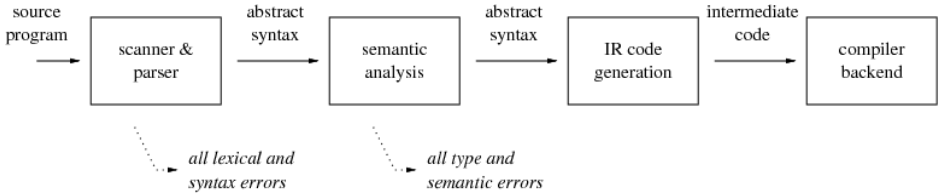
Monolithic programs are sometimes complex, unmodifiable, unmaintainable, and buggy.

Success in programming can be measured by all the programs one does *not* have to write.

This design lesson is just as important in programming-in-the-small as in programming-in-the-large.

This design lesson is important in all programming millieu: one-off scripting, system architecture, and so on.

Big problems are solved by breaking them down into small problems. Compiler: preprocess, translate, code generation, assembly, link.



Graphics pipeline or rendering pipeline: transformation, clipping, texturing.

Moreover, multiple big problems can be solved by combining well-design subproblems. Good design demands finding the best subproblems.

Example

```
find . -name '*.tex' | xargs grep -w -e 'title'
```

Find the lines containing the word title in all the TEX files.

Pipes

One of the most valuable tools every written: `grep` release in 1974 (49 years ago).

Pipes

```
sed -e 's#//.*$##' < Program.java | \
   indent --tab=3 | nl > Listing.txt
```

Take a Java program, strip the comments, indent, and number the lines.

Unix Core Utilities

Very common programs used as building blocks: `cat`, `awk`, `sed`, `grep`, `sort`, `tr`, etc.
Other useful programs: `wget`, `od`, `nl`, `cut`, `paste`, `find`, `xargs`, `tee`, etc.

A Unix programmer is one that accomplishes thousands of tasks without writing a single line of code.

`find` is useful and has many options:

```
find . -name '*.tex'

find . -name 'project* -type f -ls

find / -name 'file' -type f

find local /tmp -name 'dir' -type d -print

find / -name 'file' \& grep -v "Permission denied"

find . \( -name '*.jsp' -o -name '*.java' \) -type f -ls

find /var/ftp/mp2 -name '*.pm3' -type f -exec chmod 644 {} \;
```

xargs

`find` and `xargs` are useful and have many options:

```
find . -name '*foo*' | xargs grep bar
```

```
grep bar `find . -name '*foo*'`
```

```
find . -name '*~' -print0 | xargs -0 rm
```

```
find . -name '*foo*' -print0 | xargs -0 -I files mv files /tmp/trash
```

A Task: Histogram

1. identify words
    1. delete non-letters
    2. ignore case
    3. put each word on one line.
        1. separate words into lines by white space
        2. discard blank lines
2. count unique words
    1. group same words together
    2. count of each group
3. keep data
    1. store histogram in file
    2. give a count of unique words

Pipe Chain

Nice snake chain, man!

Pipes: Another Example

```
tr -d '?"\!:,*();><' < text | \
   tr 'A-Z' 'a-z' | \
      tr ' \t/.' '\n' | \
         sed '/^$/d' | \
            sort | uniq -c | \
               sort -rn | \
                 tee histogram | wc -l
```

Take a file and make a histogram of the words.

Pipes/Data Preparation

```
wget -O - http://www.ll.mit.edu/outside.tcpdump.gz | \
  gzip -d | \
    tcpdump -q -X -r - 'tcp and ip[0]&0x0f<6' | \
      grep -v -e '^0x..[2-9a-f]' | \
        gawk '/^..:..:../{T=substr($0,0,15)} ...' | \
          sed -n -e 50000,52000p | \
            tr -d " " | tr "-" " " | \
              nl -w 3 "-s " | \
                bzip -9 > data.bz
```

1. obtain the data over WWW
2. uncompress the gzip file
3. read tcpdump file
4. exclude some packet data
5. format data on one line
6. select just 2,000 packets
7. modify spacing
8. number the lines
9. compress the results

Streams

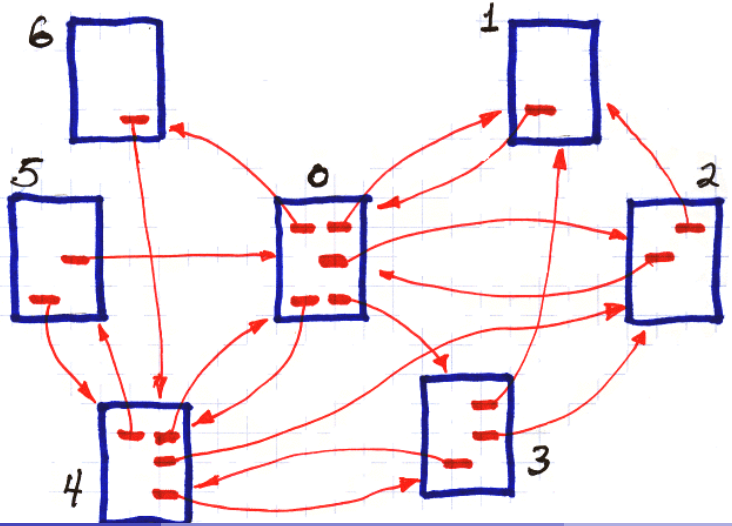
1. CopyText.java
2. CopyTextFile.java
3. GzipTextFile.java

Try

```
java CopyText < input.txt | java CopyText > output.txt
```

Example: Page Rank

Start with a graph ...



Example: Page Rank

```
> cat wiki2.txt
      7
      1 2  1 3  1 4   1 5  1 7
      2 1
      3 1  3 2
      4 2  4 3  4 5
      5 1  5 3  5 4  5 6
      6 1  6 5
      7 5
```

Example: Page Rank

Add a program to create a probability, transition matrix (Markov matrix) with leap probability 0.825.

```
> java Transition 0.825 < wiki2.txt
       7 7
        0.025   0.190   0.190   0.190   0.190   0.025   0.190
        0.850   0.025   0.025   0.025   0.025   0.025   0.025
        0.438   0.438   0.025   0.025   0.025   0.025   0.025
        0.025   0.300   0.300   0.025   0.300   0.025   0.025
        0.231   0.025   0.231   0.231   0.025   0.231   0.025
        0.438   0.025   0.025   0.025   0.438   0.025   0.025
        0.025   0.025   0.025   0.025   0.850   0.025   0.025
```

Example: Page Rank

Add a program to compute the state-stead vector (the page ranks).

```
> java Transition 0.825 < wiki1.txt | java Markov 0.277 0.158 0.139
0.109 0.185 0.063 0.071
```

Endian

In *Gulliver's Travels* by Jonathan Swift published in 1726, two factions within Lilliputian society are at war over the way to break eggs—at the big end, or the little end of the egg. The Emperor commanded all his subjects to break the smaller end, but resistance by traditionalists and subsequent suppression by the government resulted in civil unrest. Thus, Swift satirizes the suppression of Catholics in his day.

> *The Emperor . . . published an Edict, commanding all his Subjects, upon great Penaltys, to break the smaller End of their Eggs. The People so highly resented this Law, that our Historys tell us there have been six Rebellions raised on that account; wherein one Emperor lost his Life, and another his Crown.*