Programming

An Introduction to Introductions

What is a Program?

- "Detailed, step-by-step set of instructions telling the computer exactly what to do." – Zelle
- A program is a solution to a problem.
- Neural Networks versus Human Brains
- Programs are written in programming languages.
  - Some languages are compiled, some interpreted, others somewhere in between.
  - Languages use grammars that are context-free.
- How to express what to do in a way that
  - Can be verified to do what you think it does
  - Can be modified to do something else if the requirements change
  - Performs in a reasonable amount of time within the resources available

Why Program?

- Perform a (repetitive) task quickly and reliably.
- Software Engineering versus Scientific Computing
- Computer Science: What processes can be described, what qualities do they have, and what can we know about them?
- Being able to program develops analytic skills and frees one from burdensome repetition. You’re too valuable for busy work.

Programing Languages

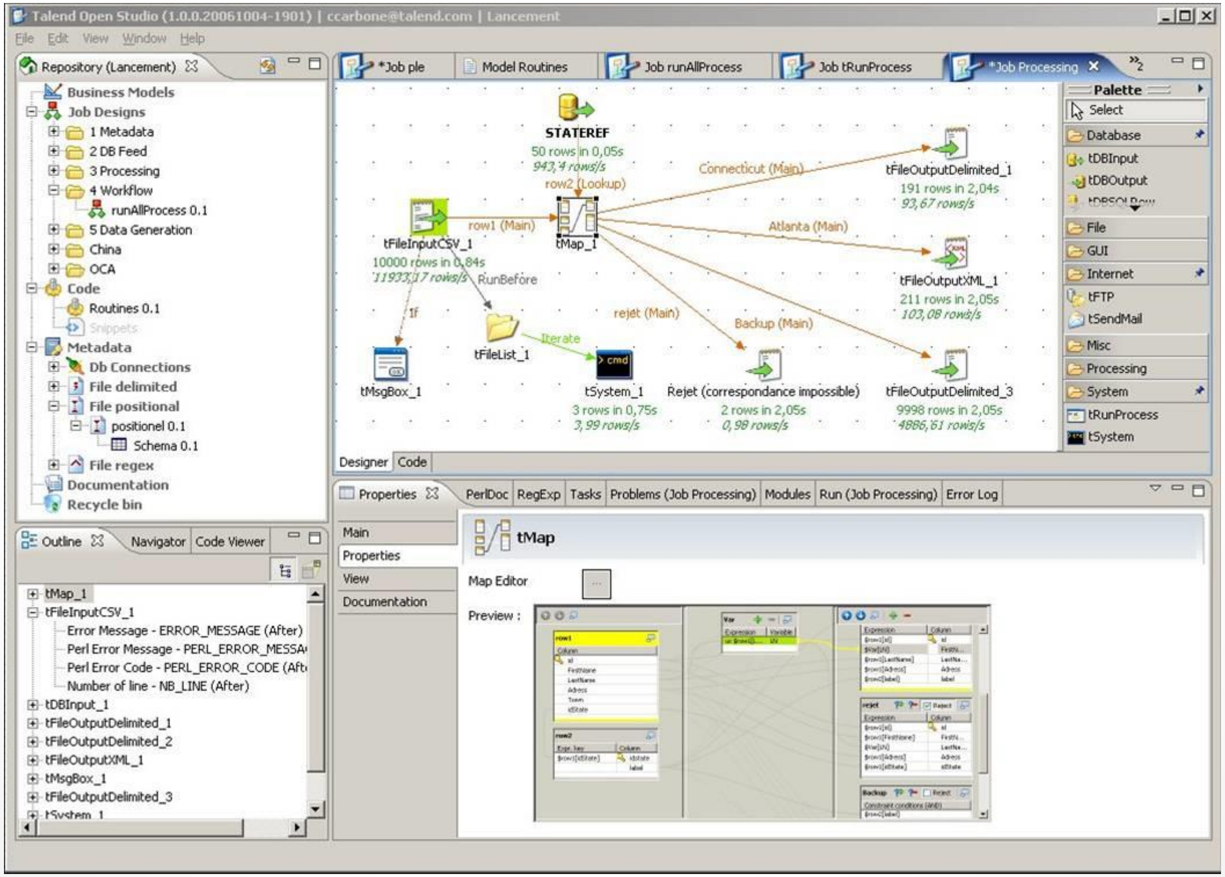
- There are many, many, many.
- Tool for the job.
- Some general, some domain specific.
- People like to argue about which is better.
- Most real-world solutions use several different languages.

Programing Languages 2

- Each language has rules – the syntax.
- Each language has idioms – efficient way of doing things specific to the language.
- Patterns are general solutions to problems commonly encountered.

Programing Languages 3

- Research into graphical representations of programs, a higher level way to interact with the code.



Anatomy of a Program

Comments

- A way of putting information into code that does not effect execution of the code.

```
# Sometimes pound symbol
// Or this
/*  or this */
% Even this maybe
```

Statements

- A unit of detail best described as a step in your program.
- Some languages use delimiters like the semicolon to indicate that one statement has ended. Python can use semicolon or a line break

```
Stmt1; Stmt2;
Stmt3;
```

Blocks

- A block is a collection of statements that share the same state. A single statement can be a block.

- In Python, blocks are contiguous areas of the same indentation. In most languages the curly braces are used to denote blocks.

```
{ Stmt1; Stmt2; } {Stmt3;}
```

Variables

- Variables are a way of labeling and storing data.
- Static versus dynamic
- Data types

```
float radius = 1.62f;
var radius = 1.62;
radius = 1.62;

String name = “Edgar”;
var name = “Edgar”;
name = “Edgar”
```

Functions

- Instructions
- Return values
- Arguments

```
function two() return 1+1;

function plusOne(x){
        return x+1;
}

plusOne(2)
```

Variables Again

- Can be a single value, or a complex instance of a data type:
  - Arrays
  - Functions
  - Objects

```
var places = array(“Here”, “There”,
“Everywhere”);

Person Edgar = new Person(“Edgar”,”Hassler”);

var f = function(t){ return(t*(t+1)); }
```

Objects

- In the object oriented (OO) paradigm we allow for the definitions of *objects* that combine *data* with *behavior*.

- Function that are attached to an object are called methods. Variables that belong to an object are called Properties.

```
Person Edgar = new Person(“Edgar”,”Hassler”);
Edgar->visitClass();
print Edgar->getPosition();
```

Scope

- The variables that can be seen by statements in the same block are called the scope.
- Outside of the block, these variables are not visible, and may be reassigned by the computer to some other variable.

```
var test = 1;
{
        var test = 2;
}
print test
```

Scope

```
def mpower(m):
     def raiseTo(n):
          return m**n

f = mpower(2)
f(4)                                  # 16
```

Control Structures

- Any non-trivial program will change its behavior based on the inputs, and control structures are how this is done.
- Most loops involve control structures that govern their execution.
  - If then
  - If else then
  - While do
  - Do until
  - Switch case
  - For
  - Foreach

Control Structures

- In Python, compare using

Python	Natural Lang.
`<`	Less than
`<=`	Less than or equal
`==`	Equal
`>=`	Greater than or equal
`>`	Greater than
`!=`	Not equal to

- `True`, `False`. Negate using `not`

Floating Point Numbers

- How we store real numbers.
- Sign, base, exponent
- Underflow and overflow

```
test = 0.1

while test < test + 0.1 do
     test = test + 0.1
end
```

31337 H4X0Rz

- The instructions that constitute your program are stored in memory.
- Variables are stored in memory.
- If careless, external data written to memory can overwrite your instructions.

```
char buf[8];
gets(buf);
fprintf(“%s\n”,buf);
return 0;
```

Concurrency

- Computers today can do several tasks at once. But our methods of programming are usually ill-suited to address this.
- A whole set of design techniques exist to address these issues.

```
from threading import Thread, Lock

mutex = Lock()

def processData(data):
    mutex.acquire()
    try:
        print('Do some stuff')
    finally:
        mutex.release()

while True:
    t = Thread(target = processData, args = (some_data,))
    t.start()
```

Massive Parallelism

- For supercomputing clusters and GPU computing, we have very many threads that we can run in parallel.

```
sapply(initalConditions,function(start){
     … code to be run many times here …
})
```

Programming Languages

- Fortran, C, C++
- Java
- Python, PHP, Ruby

Practical Concerns

Get Python

- Is it already on your computer? Try `python`.
- Download it from http://www.python.org/download/

Integrated Development Environment (IDE)

- And IDE helps you program.
- Eclipse -> PyDEV, or something else.

Try it out!

- http://www.learnstreet.com/lessons/study/python
- http://wiki.python.org/moin/BeginnersGuide/NonProgrammers

```
# convert.py
# A program to convert Celsius temps to Fahrenheit
def main():
        celsius = input("What is the Celsius temperature? ")
        fahrenheit = 9.0 / 5.0 * celsius + 32
        print "The temperature is", fahrenheit, "degrees 
Fahrenheit."
main()
```

Try it out!

```
# quadratic4.py
import math
def main():
        print "This program finds the real solutions to a quadratic\n"
        a, b, c = input("Please enter the coefficients (a, b, c): ")
        discrim = b * b - 4 * a * c
        if discrim < 0:
                print "\nThe equation has no real roots!"
        elif discrim == 0:
                root = -b / (2 * a)
                print "\nThere is a double root at", root
        else:
                discRoot = math.sqrt(b * b - 4 * a * c)
                root1 = (-b + discRoot) / (2 * a)
                root2 = (-b - discRoot) / (2 * a)
                print "\nThe solutions are:", root1, root2
```

A Little More Theory

Data Structures

- Commonly used ways of organizing data and behavior.
- Examples:
  - Queue – FIFO
  - Stack – LIFO
  - Linked List – Single and Double
  - Trees – for sorting and accessing
  - Many more

Object Oriented Principles

- Hierarchy – objects belong to classes organized into hierarchy. Ad hoc ontology.
- Establishing these relationships helps build an understanding of the abstract qualities shared by different parts of your problem.
- A decomposition of a problem into parts (separation of concerns, encapsulation)
- Allows us to change one part of the program with a guarantee the rest will function (modularity). This also helps with re-use.

Design Patterns

- Model, View, Controller
  - Model – Encapsulates the data and its behavior
  - View – Describe various ways to present the model data
  - Controller - Handle requests and decide on models and views.
- Command
  - Let an object represent a command and its state.
- Lazy Loading
  - Only use resources when you need them.
- Database Patterns
  - Active record – Object represents live copy of database data
  - Data mapper – A third party maps data between models and database
  - Table module – A single object handles all database data

Aspect Oriented Princip.

- Crosscutting concerns
- Example:
  - Several controllers require that the connection be secure – Aspect!
  - Several threads want to wait until the stack of jobs is empty – Aspect!

Abstract Ideas

- Once and Only Once
  - All code must appear in only one place. No copy pasting!
  - Sometimes called Don't Repeat Yourself (DRY) principle.
- Separate the What from the How
  - A method should comprise one how, or two or more whats.
  - "What" is a delegation to another method with a meaningful name.
  - "How" is a method of doing one thing.
- The What but not the Why
  - Code is a blueprint for what to do. An architect provides blueprints for a house and not the reason for certain features. Code and architecture are separate concerns.
  - Use comments liberally to document the "Why".
- Everything should be testable.
  - Also those tests should be automatable.
- You Aren't Gonna Need It (YAGNI) (KISS corollary)
  - Only write code for things when you need it. Prevents over-engineered solutions.

A Comment on Time

- One of the hardest types of data to work with is time. Not only does it vary relative to physical location, but it has events that have non-standard periodicity
  - New years day – 1st of each year
  - Labor day – 1st Monday of September
  - There are 52 weeks in a year, most years
  - A week starts on Sunday, unless it starts on Monday
  - There is a leap day every 4 years, except every 100 years, except every 400 years.
  - Easter – no one knows.
  - Some cultures rely on lunar calendars