Scripting Languages

bash, Ruby, Python, Pearl, and PHP are examples of scripting languages. They are designed for small tasks that involve coordination or communication with other programs.

Common properties:
- Interpreted, with dynamic typing
- Emphasis on expressivity and ease of programming over efficiency
- Allows multiple paradigms (functional, imperative, object-oriented)
- Built-in string handling, data types
- Extensive “shortcut” syntactic constructs

Scripting example: Prime generation in Python

```python
def PrimeGen():
    for p in itertools.count(2):
        if (reduce(lambda a,b: a and b, \map(lambda i: p%i != 0, range(2,p)),True)):
            yield p

for p in PrimeGen():
    if (p < 100): print p
    else: break
```

Generators

Sometimes a function computes multiple values as it goes along. An iterator created automatically from such a function is called a generator.

Simpler (related) Python example:

```python
def factors(n):
    for i in range(2,n):
        if (n % i == 0): yield i
```
The Need for Generic Code

A *function* is an abstraction of similar behavior with *different values*.

Generic code takes this to the next level, by abstracting similar functions (or classes) with *different types*.

Most common usages:
- Basic functions: min/max, sorting
- Collections: vector, linked list, hash table, etc.

Genericity in Scheme

In Scheme and other languages with *run-time type checking*, writing generic functions is (mostly) trivial.

Generic minimum function:

```scheme
(define (minimum a b)
  (if (<= a b) a b))
```

Generic binary tree structure:

```scheme
(lambda (command)
  (cond [(symbol=? command 'left) left]
        [(symbol=? command 'right) right]
        [(symbol=? command 'root) ele]))
```

```scheme
(define BST (make-bt 4
               (make-bt 2
                    (make-bt 1 null null)
                    (make-bt 3 null null)))
               (make-bt 6
                    (make-bt 5 null null)
                    (make-bt 7 null null)))
```

Genericity in C++

Old School (C style)
- Use *function-like macros* to code-generate every possibility.
- Types to be used in generic functions/classes must be explicitly specified.

Templates (C++ style)
- Built into the language; don't rely on preprocessor
- Compiler does code generation, similar to macros
- Types to be used are determined *implicitly* at compile-time
- *Separate compilation* becomes difficult or impossible.
C++ Genericity with Macros

```cpp
#define WRITE_LL_CLASS(T) 

class Node_ ## T { 
public: 
T data; 
Node_ ## T * next; 
Node_ ## T (T d, Node_ ## T * n) : data(d), next(n) { } 

T printAndSum() { 
cout << data << endl; 
if (next == NULL) return data; 
else return data + next->printAndSum(); 
}
};

WRITE_LL_CLASS(float)
WRITE_LL_CLASS(int)
```

```cpp
int main() {
Node_float * fhead = NULL;
Node_int * ihead = NULL;
// ... fill the lists with some input

cout << "Floating sum: " << fhead->printAndSum() << endl << endl;
cout << "Int sum: " << ihead->printAndSum() << endl << endl;
}
```

C++ Genericity with Templates

```cpp
template <class T>
class Node {
public:
T data;
Node<T> * next;
Node(T d, Node<T> * n) : data(d), next(n) { } 

T printAndSum() { 
cout << data << endl; 
if (next == NULL) return data; 
else return data + next->printAndSum(); 
}
};
```

```cpp
int main() {
Node<float> * fhead = NULL;
Node<int> * ihead = NULL;
// ... fill the lists with some input

cout << "Floating sum: " << fhead->printAndSum() << endl << endl;
cout << "Int sum: " << ihead->printAndSum() << endl << endl;
return 0;
}
```

Genericity in Java

Old School (Java ≤ 1.4)
- Use abstract base classes/interfaces like Object
- Make extensive use of polymorphism
- Lots of upcasting and downcasting

Generics (Java ≥ 5)
- Similar syntax to C++ templates
- Compiler checks type safety then removes generic type information
- Up/downcasting still performed, implicitly
- Generics are only syntactic sugar
Manual Genericity in Java

```java
interface Sum {
    void add(Number x);
}

class FloatSum implements Sum {
    float val = 0;
    public void add(Number x) {
        val += ((Float)x).floatValue();
    }
    public String toString() { return String.valueOf(val); }
}

class IntSum implements Sum {
    int val = 0;
    public void add(Number x) {
        val += ((Integer)x).intValue();
    }
    public String toString() { return String.valueOf(val); }
}
```

Java 5 Generics

```java
interface Sum<T> {
    void add(T x);
}

class FloatSum implements Sum<Float> {
    float val = 0;
    public void add(Float x) {
        val += x.floatValue();
    }
    public String toString() { return String.valueOf(val); }
}

class IntSum implements Sum<Integer> {
    int val = 0;
    public void add(Integer x) {
        val += x.intValue();
    }
    public String toString() { return String.valueOf(val); }
}
```
```java
class LLNew<T> {
    T data;
    LLNew<T> next;
    LLNew(T d, LLNew<T> n) { data = d; next = n; }
}

Sum<T> printAndSum(Sum<T> summer) {
    System.out.println(data);
    summer.add(data);
    if (next != null) next.printAndSum(summer);
    return summer;
}

public static void main(String[] args) {
    LLNew<Float> flist = null;
    LLNew<Integer> ilist = null;
    // ... fill the lists with some input
    System.out.println("Floating sum: "+
        flist.printAndSum(new FloatSum()) + 
        "\n");
    System.out.println("Integer sum: "+
        ilist.printAndSum(new IntSum()) + 
        "\n");
}
```

Trade-Offs in Generics

- **No declared types**
  - No enforced notion of “list of integers” etc.
  - Requires dynamic typing; slower

- **Code Generation (C++ templates)**
  - Can result in (combinatorial!) code explosion
  - Very powerful and general, but somewhat unintuitive

- **Code Annotation (Java 5 generics)**
  - Syntactic sugar; extensive run-time casting results
  - Types not known to the program at runtime — eliminates some capabilities

Class outcomes

You should know:
- What a scripting language is
- When/why scripting languages are used
- What a generator is
- What a generic class/function is
- Genericity in dynamically-typed languages
- How genericity works in C++ and Java
- Trade-offs in getting genericity in programming languages