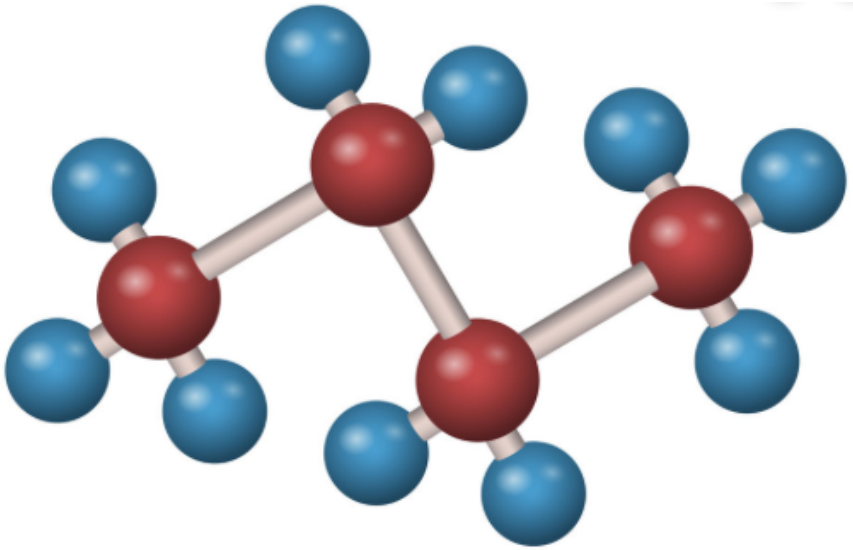
Part I

The Structure of Programs

The Elements

- Question: What are the basic elements that make up a computer program?
- Can you deconstruct a program into a basic collection of objects?
- In essence: A data model



Primitives

- There are primitive values

```
34                  # Integer
3.4                 # Float
'T'                 # Character
True                # Boolean
```

- The primitives are the most basic things
- Indivisible
- The foundation of all else

Types

- There is usually an underlying type system

```
int
float
char
bool
```

- Values have an associated "type"
- Required to map operations onto actual hardware (e.g., integer operations vs floating point operations).

Names

- You can name things

```
var r = 2.0;
const pi = 3.14159;
var area = pi * r * r;
```

- You don't hardcode values, use a name
- New names often introduced by "declarations"
- Example: "var", "const", etc.
- Naming -> "Abstraction"

Expressions

- There are operators (often on hardware)

```
3 + 4 * 5
tau = 2 * pi
```

- And rules for evaluation order (left-right)

```
3 - 4 - 5          # -> (3 - 4) - 5
```

- And rules for precedence (from math)

```
3 - 4 * 5          # -> 3 - (4 * 5)
```

- An expression always produces a value

Assignment

- Computers have memory
- Load/store operations

```
3 + x;          // Value is read from "x"
x = 4 + 5;      // Value is stored in "x"
```

- A "storage location" is a complex concept

```
x               // Simple value
x[n]            // Indexing (arrays)
x.attr          // Attribute (structures)
```

- Locations can appear on either side of =

```
x[n] = y.attr;
```

- Also: Mutable vs Immutable

Control Flow

- You can make decisions

```
if a > b {
    m = a;
} else {
    m = b;
}
```

- And perform repeated operations

```
while n > 0 {
    print('T-minus', n);
    n = n - 1;
}
```

Functions/Procedures

- Defining a function

```
func square(x float) float {
    return x*x;
}
```

- Applying a function (produces a value)

```
3 + square(10)
```

Data Structures/Types

- There may be more complex data types

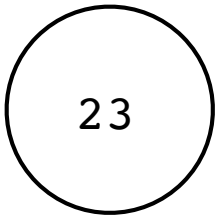
```
int [10]          // Arrays/Vectors

struct Point {    // Records (multiple values at once)
    x int;
    y int;
}

enum Choice {     // Enumerations (one value chosen)
    Yes;
    No;
}
```

- Arrays, structures, enums, classes, etc.
- Built upon the more primitive types.

Problem: Representation

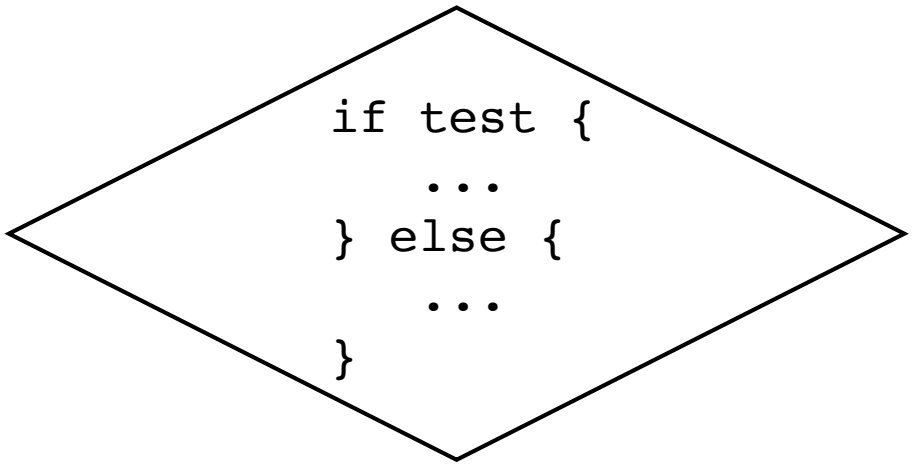
- How do you represent a computer program as a a proper data structure?
- Not as text, but as concrete objects
- Like in a database
- Or as a diagram you would draw

Problem: Representation



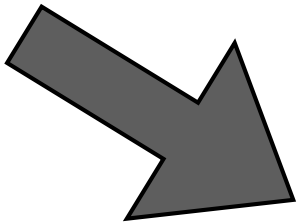

```
const pi = 3.14159;
```

Program Elements

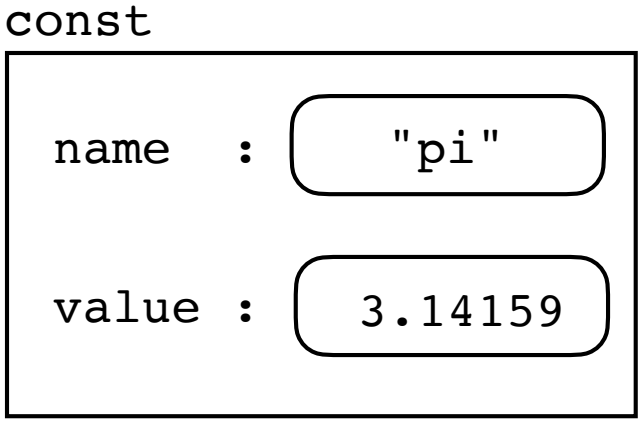


Elements Have Parts

```
const pi = 3.14159;
```

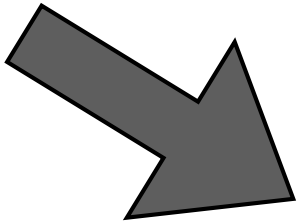


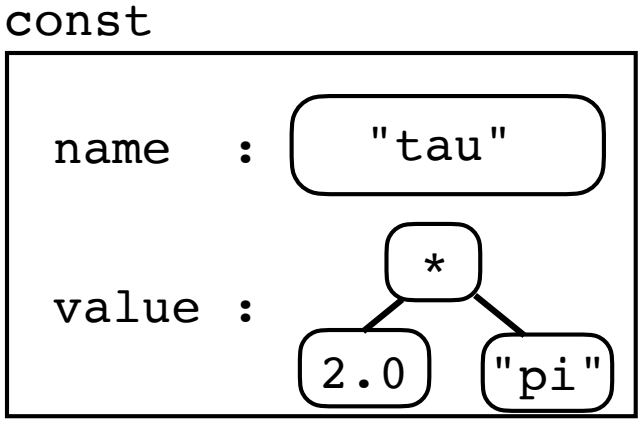
Program elements are built from more basic primitives (names, types, etc.)



Parts can be Complex

```
const tau = 2.0 * pi;
```

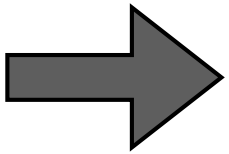


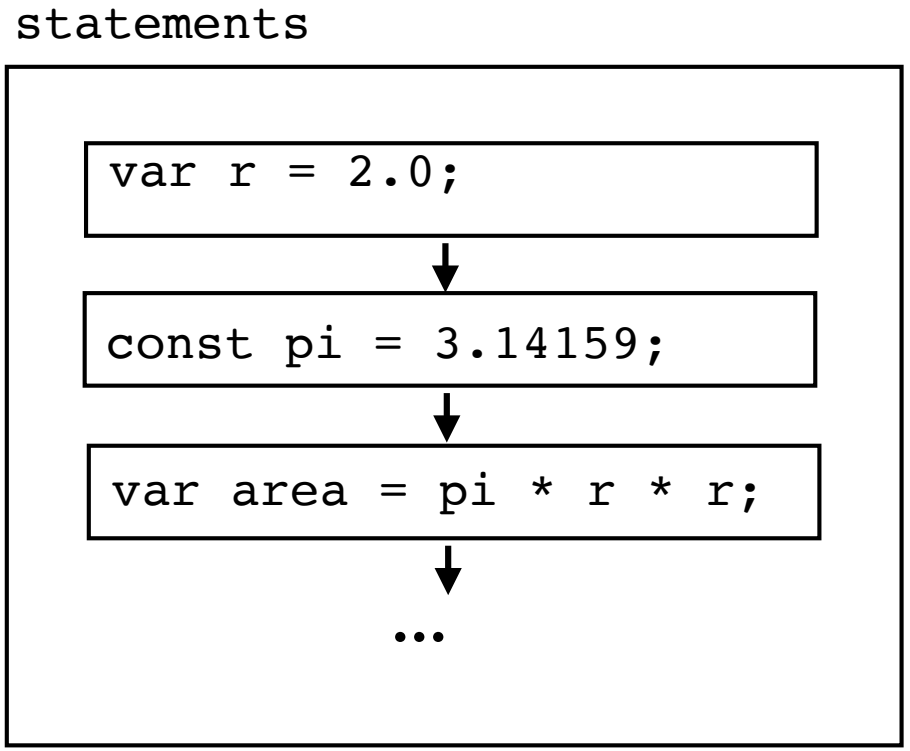


There is often a high degree of nesting (elements contained within other elemnts)

Collections

```
var r = 2.0;
const pi = 3.14159;
var area = pi * r * r;
```

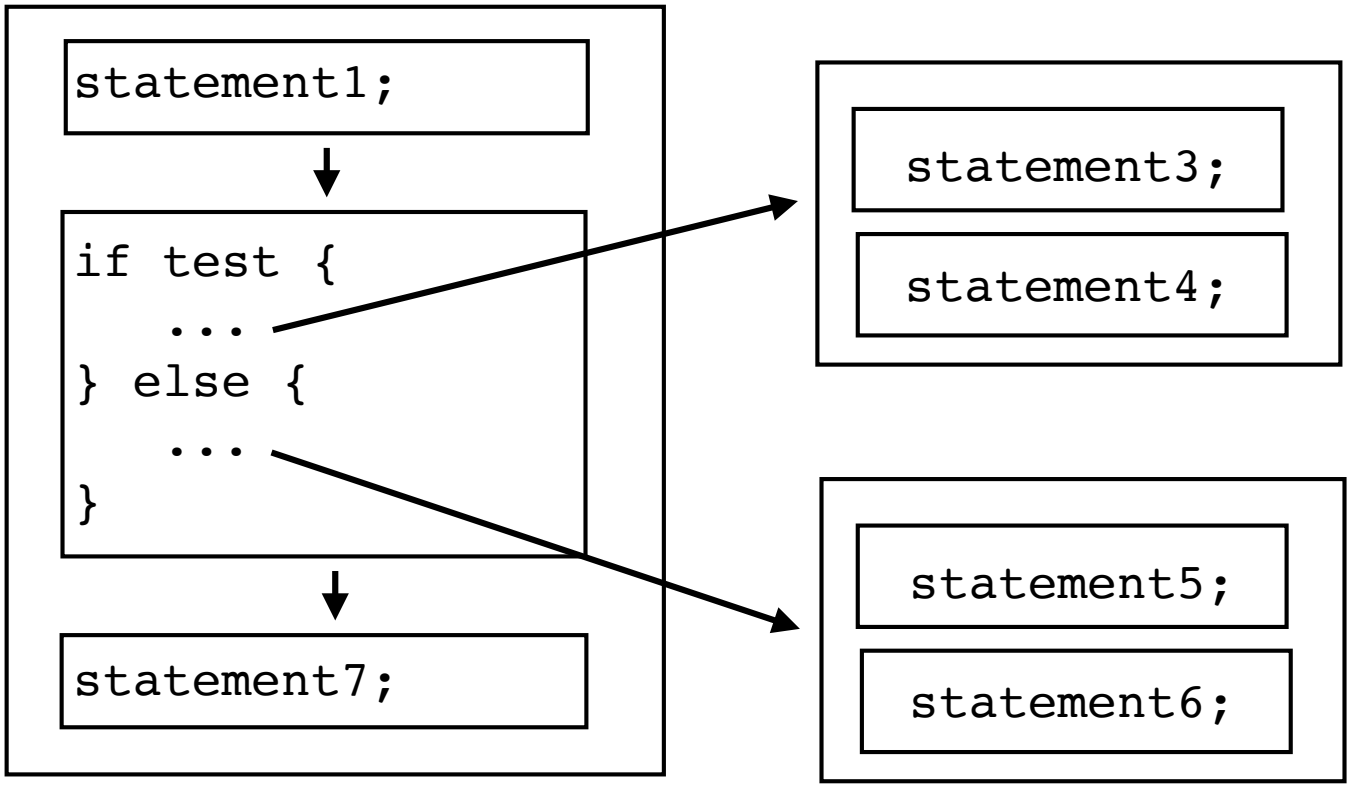




Multiple statements
Multiple structure fields
Multiple function arguments
Many multiples!

Nesting

```
statement1;
if test {
   statement3;
   statement4;
} else {
   statement5;
   statement6;
}
statement7;
```



Programs are filled with nested structure

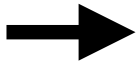
Programs as Objects

- Program elements can be defined by classes

```
23
```

```
class Integer:
    def __init__(self, value):
        self.value = value
```

```
location = value;
```



```
class Assignment:
    def __init__(self, location, value):
        self.location = location
        self.value = value
```

```
left + right;
```

```
class BinOp:
    def __init__(self, op, left, right):
        self.op = op
        self.left = left
        self.right = right
```

Programs as Objects

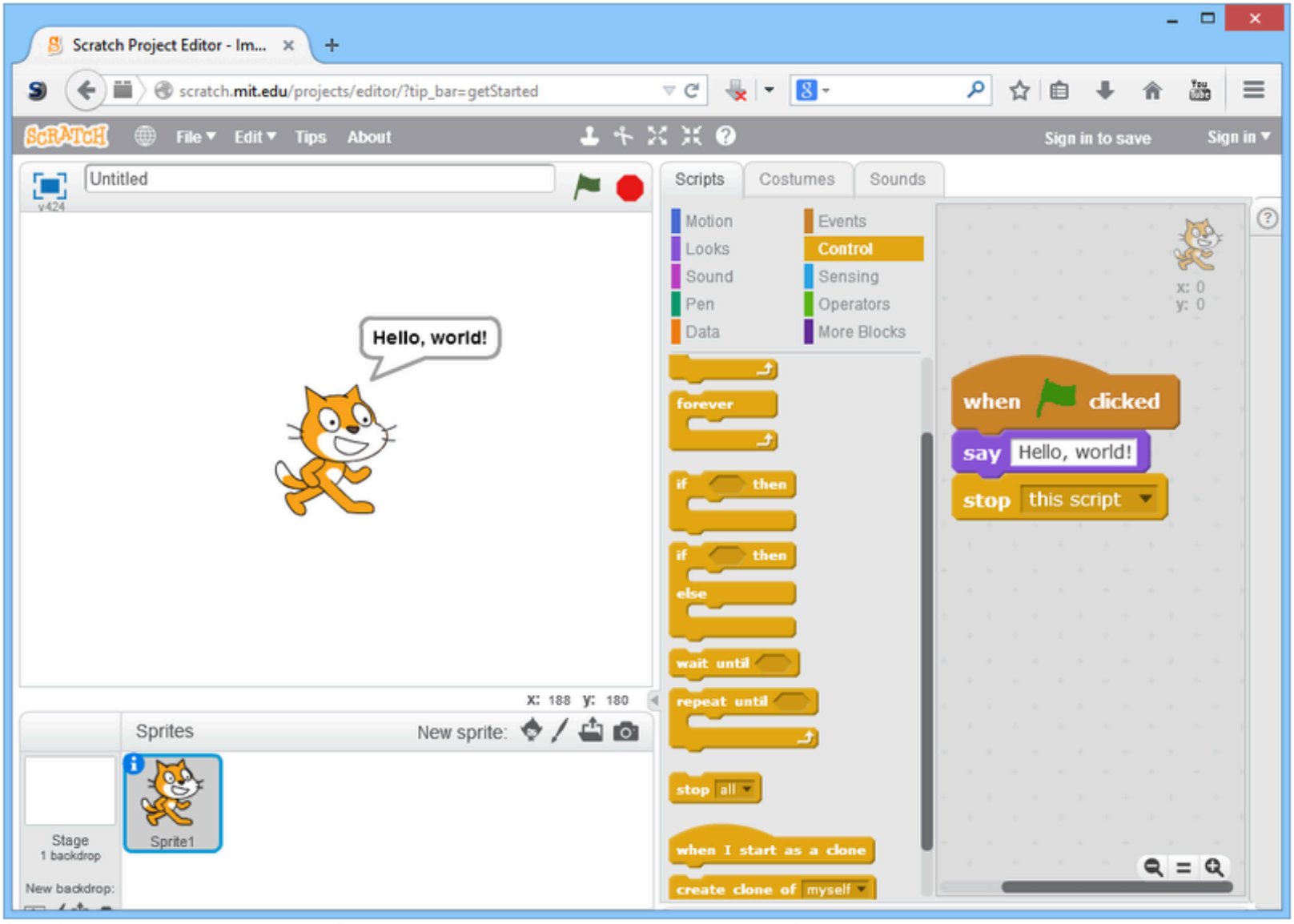
- Programs representation example:

```
x = 23 + 42;

Assignment(
  NamedLocation('x'),
  BinOp('+', Integer(23), Integer(42))
)
```

- Commentary: A major part of writing a compiler is in designing and building the data model. It directly reflects the structure and features of the language that's being compiled.
- Sometimes known as Abstract Syntax Tree (AST)

Commentary

- Programs are not necessarily "text"



Commentary

- A structurally correct program is not necessarily a correct program

```
const pi = "three";
pi = pi + .14159;
```

- The syntax might be fine
- The meaning might be gibberish
- Don't confuse program semantics with program structure. They are two different problems.
- Right now: Structure. Not behavior.

Project 1

Find the files

- `wabbit/model.py`
- `script_models.py`

Follow the instructions inside (with guidance)