Part I

The Structure of Programs

A Program

```
/* Recursive Fibonacci numbers */

func fib(n int) int {
    if n < 2 {
        return 1;
    } else {
        return fib(n-1) + fib(n-2);
    }
    return 0;
}

const LAST = 30;

func main() int {
    var n = 0;
    while n < LAST {
        print fib(n);
        n = n + 1;
    }
    return 0;
}
```

A Program

```
/* Recursive Fibonacci numbers */

func fib(n int) int {
    if n < 2 {
        return 1;
    } else {
        return fib(n-1) + fib(n-2);
    }
    return 0;
}

const LAST = 30;

func main() int {
    var n = 0;
    while n < LAST {
        print fib(n);
        n = n + 1;
    }
    return 0;
}
```

Q: What do you see?

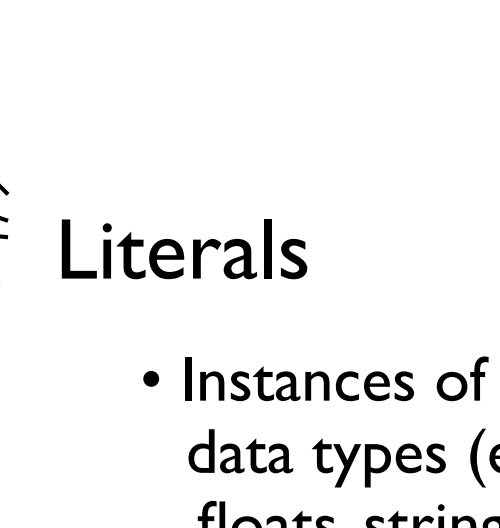
A Program

```
/* Recursive Fibonacci numbers */

func fib(n int) int {
    if n < 2 {
        return 1;
    } else {
        return fib(n-1) + fib(n-2);
    }
    return 0;
}

const LAST = 30;

func main() int {
    var n = 0;
    while n < LAST {
        print fib(n);
        n = n + 1;
    }
    return 0;
}
```



Literals

- Instances of primitive data types (e.g. ints, floats, strings, etc.)

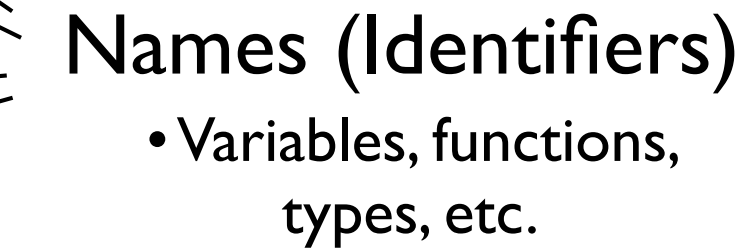
A Program

```
/* Recursive Fibonacci numbers */

func fib(n int) int {
    if n < 2 {
        return 1;
    } else {
        return fib(n-1) + fib(n-2);
    }
    return 0;
}

const LAST = 30;

func main() int {
    var n = 0;
    while n < LAST {
        print fib(n);
        n = n + 1;
    }
    return 0;
}
```



A Program

```
/* Recursive Fibonacci numbers */

func fib(n int) int {
    if n < 2 {
        return 1;
    } else {
        return fib(n-1) + fib(n-2);
    }
    return 0;
}

const LAST = 30;

func main() int {
    var n = 0;
    while n < LAST {
        print fib(n);
        n = n + 1;
    }
    return 0;
}
```


Expressions

- Represent a value
- Operators (+, -, *, ..)
- Combinations of operators

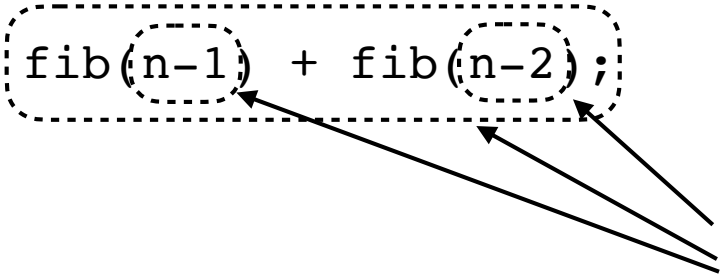
A Program

```
/* Recursive Fibonacci numbers */

func fib(n int) int {
    if n < 2 {
        return 1;
    } else {
        return fib(n-1) + fib(n-2);
    }
    return 0;
}

const LAST = 30;

func main() int {
    var n = 0;
    while n < LAST {
        print fib(n);
        n = n + 1;
    }
    return 0;
}
```



Nested Expressions

- Expressions are built from other expressions. It can be arbitrarily complicated

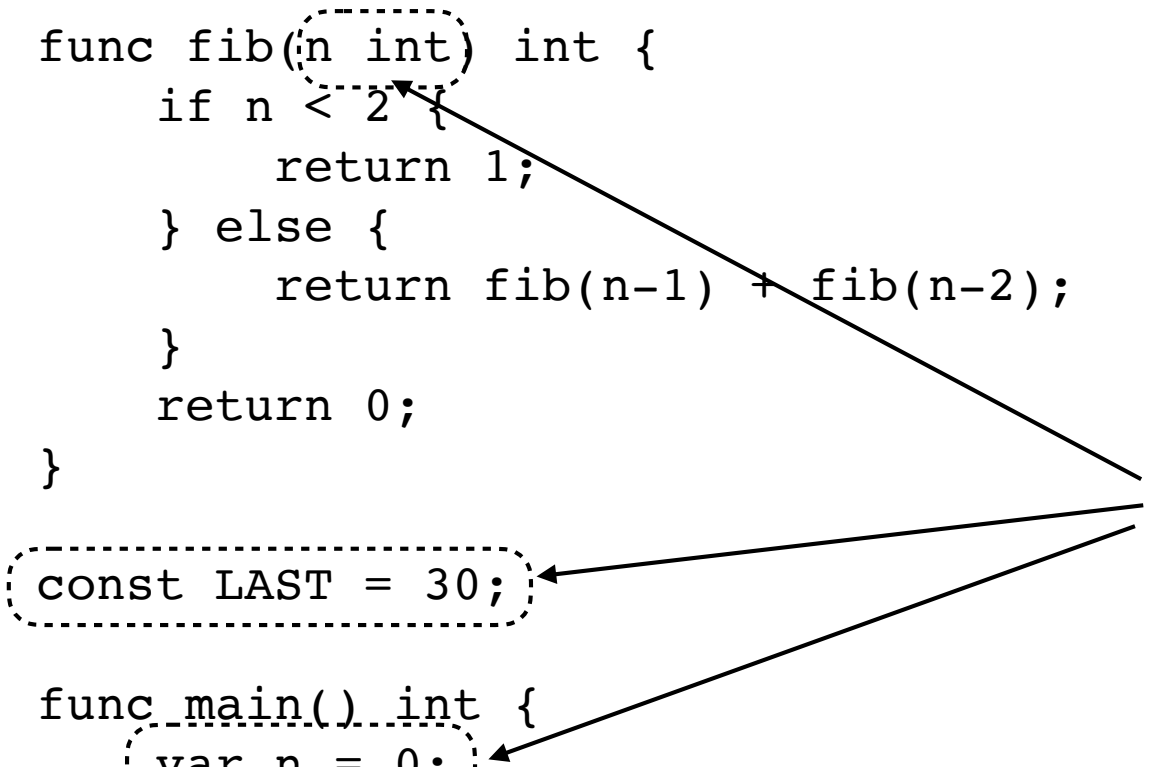
A Program

```
/* Recursive Fibonacci numbers */

func fib(n int) int {
    if n < 2 {
        return 1;
    } else {
        return fib(n-1) + fib(n-2);
    }
    return 0;
}

const LAST = 30;

func main() int {
    var n = 0;
    while n < LAST {
        print fib(n);
        n = n + 1;
    }
    return 0;
}
```



Definitions/Declarations

- Define existence of names

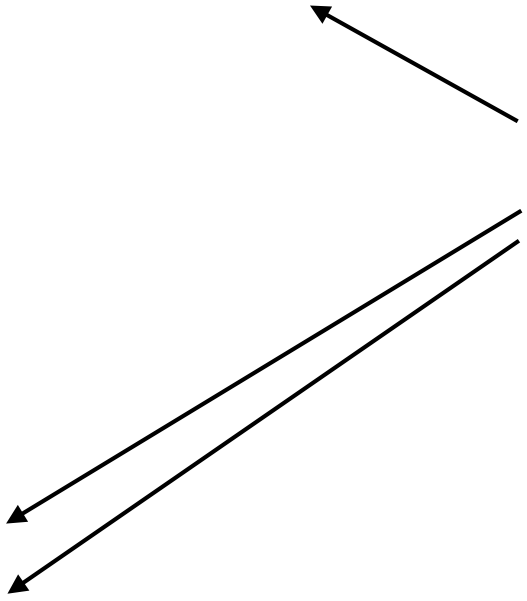
A Program

```
/* Recursive Fibonacci numbers */

func fib(n int) int {
    if n < 2 {
        return 1;
    } else {
        return fib(n-1) + fib(n-2);
    }
    return 0;
}

const LAST = 30;

func main() int {
    var n = 0;
    while n < LAST {
        print fib(n);
        n = n + 1;
    }
    return 0;
}
```



Statements

- Step-by-step sequencing
- Usually have a side effect
- Example: assignment, printing

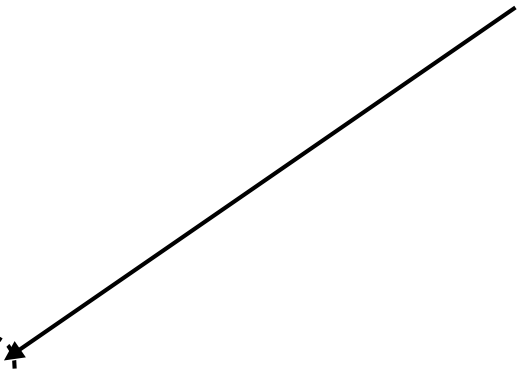
A Program

```
/* Recursive Fibonacci numbers */

func fib(n int) int {
    if n < 2 {
        return 1;
    } else {
        return fib(n-1) + fib(n-2);
    }
    return 0;
}

const LAST = 30;

func main() int {
    var n = 0;
    while n < LAST {
        print fib(n);
        n = n + 1;
    }
    return 0;
}
```



Statement Blocks

- Multiple statements
- Grouped together

A Program

```
/* Recursive Fibonacci numbers */

func fib(n int) int {
    if n < 2 {
        return 1;
    } else {
        return fib(n-1) + fib(n-2);
    }
    return 0;
}

const LAST = 30;

func main() int {
    var n = 0;
    while n < LAST {
        print fib(n);
        n = n + 1;
    }
    return 0;
}
```

Nested Statements

- Statements inside statements

Problem: Representation

- How do you represent a computer program as a a proper data structure?
- Not as text, but as concrete objects
- Something that you can work with as data

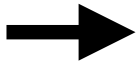
Programs as Objects

- Program elements can be defined by structures

```
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

- Example:

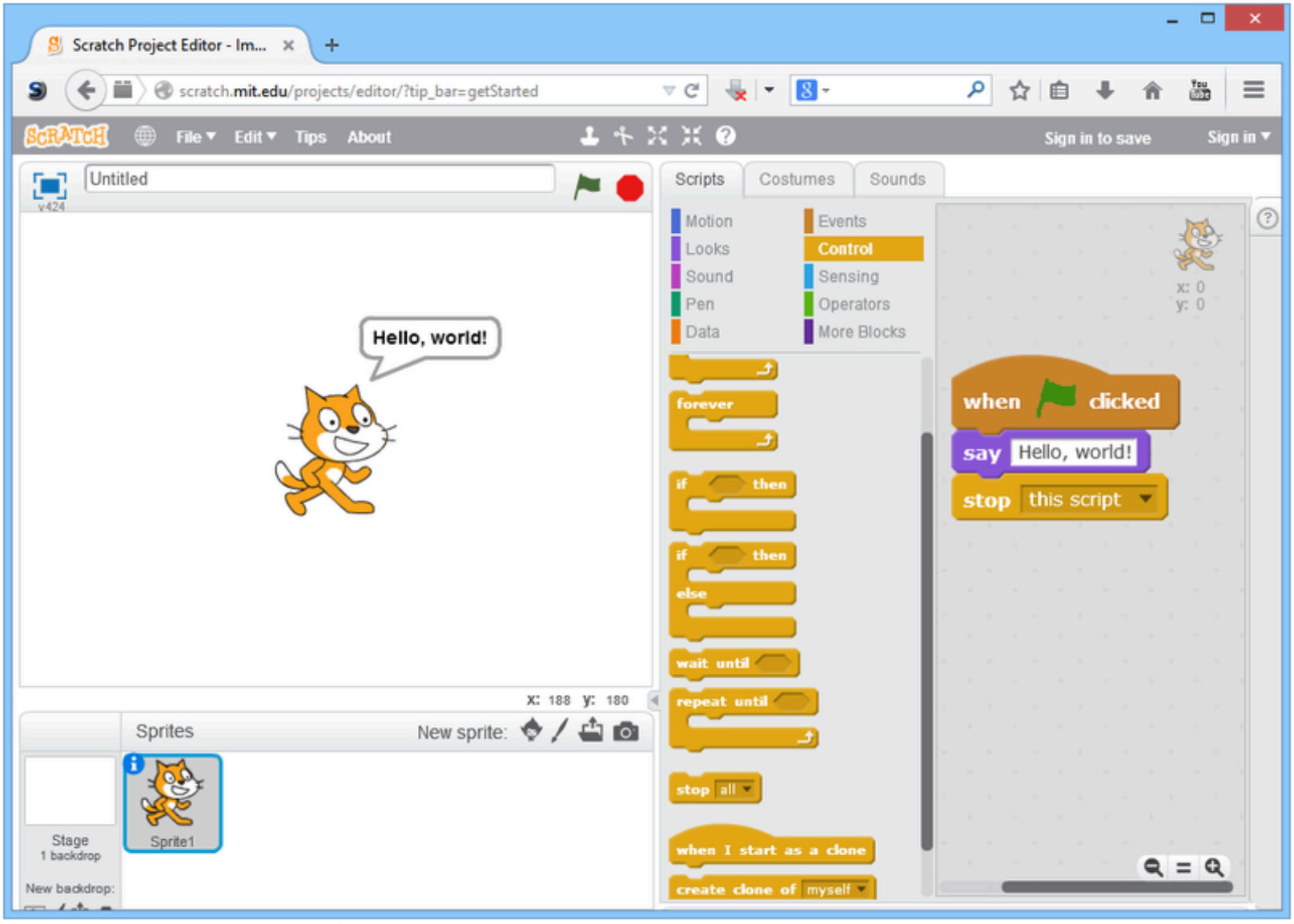
```
x = 23 + 42;
```

```
Assignment(
  Name('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 features of the language that's being compiled and how parts are composed.
- Sometimes known as Abstract Syntax Tree (AST)

Commentary

- Programs are not necessarily "text"



Commentary

- A structurally correct program is not necessarily a correct running program

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

- "correct looking" != "correct running"
- Don't confuse program semantics with program syntax. They are two different problems.
- Right now: Structure. Not behavior.

Project 1

Find the files

- `wabbit/model.py`
- `test_models.py`

Follow the instructions inside (with guidance)