# Algorithms and Programming I

Lecture#12

Spring 2015

# Think Python


## How to Think Like a Computer Scientist

By :Allen Downey

# Installing Python

- Follow the instructions on installing Python and IDLE on your own computer on the Python.org website :

www.python.org

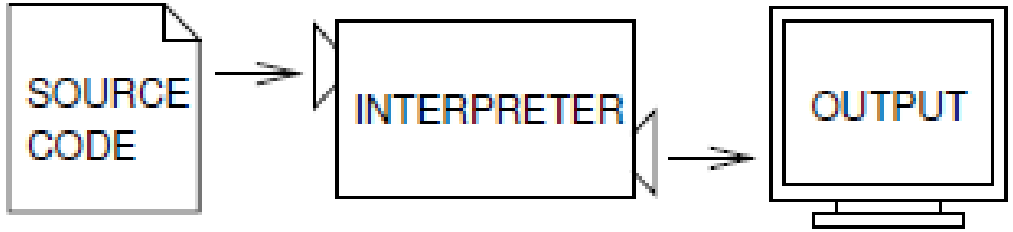
# The Python Programming Language

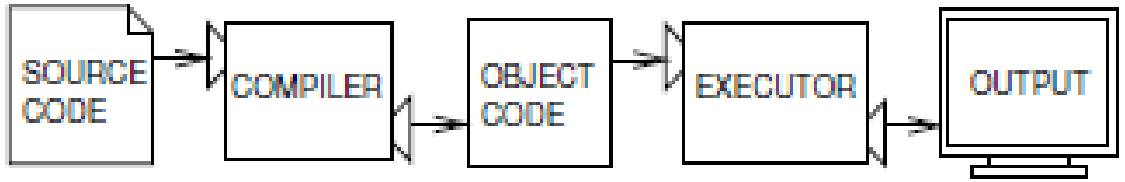
- High_ Level language vs Low_ level Language “ machine language”
  Close to the machine!
- General vs Targeted
- Interpreted vs Compiled

# The Python Programming Language…

- Programs process high-level languages into low-level languages:
  1. **Interpreters**
  2. **Compilers**

  The high _level program is called ***source code***, and the translated program is called the ***object code*** or executable.



# The Python Programming Language…

- Python is considered an interpreted language because Python programs are executed by an interpreter.
- There are two ways to use the interpreter:
  - **Interactive mode**
    - >>> 1+1

      2

      The chevron, >>> is the prompt the interpreter uses to indicate that it is ready.
  - **Script mode**
    - Alternatively, you can store code in a file and use the interpreter to execute the contents of the file, which is called a script. By convention, Python scripts have names that end with .py.
    - To execute the scrip, you have to tell the interpreter the name of the file.

# What is a program?

- A program is a sequence of instructions that specifies how to perform a computation. the computation might be something mathematical , such as solving a system of equations, or finding the roots of a polynomial, but it can also be a symbolic computation, such as searching and replacing text in a document or compiling a program.
- The details look different in different languages, but a few basic instructions appear in just about  every language:
    - Input
    - Output
    - Math
    - Conditional execution
    - repetition

# What is Debugging

- programming errors are called **bugs**
- the process of tracking them down is called **debugging**.
- Three kind of errors can occur in a program:
    1. Syntax Errors
    2. Runtime errors
    3. Semantic errors

# Syntax Error

- Python can only execute a program if the syntax is correct; otherwise, the interpreter displays an error message.
- **Syntax** refers to the structure of a program and the rules about that structure. Legal expression of the language!
  - For example, parentheses have to come in matching pairs, so (1 + 2) is legal, but 8) is a **syntax error**.
    - Lots of help build in python- python checks it for you . So don’t turn an assignment with syntax error or you will get a zero.

# Runtime errors

- The second type of error is a Runtime error, so called because the error does not appear until after the program has started running.

# Semantic Errors

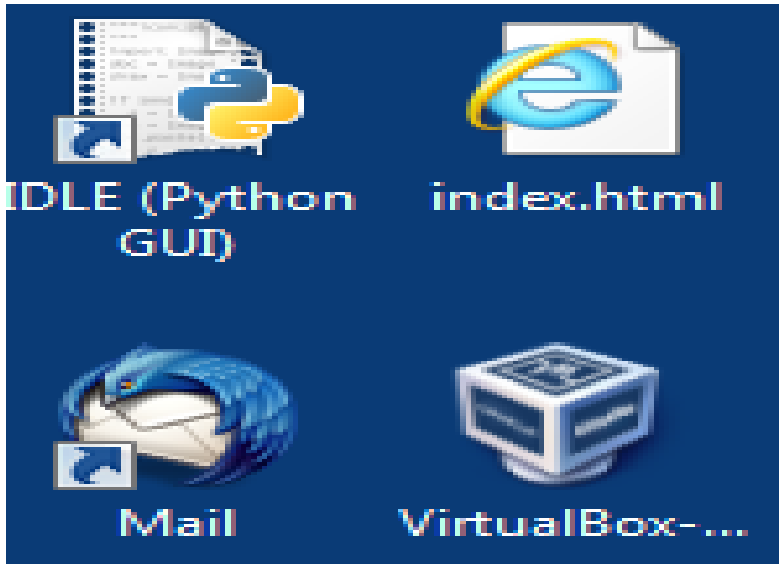
- The third type of error is the **semantic error**. If there is a semantic error in your program, it will run successfully and the computer will not generate any error messages, but it will not do the right thing!!
- Identifying semantic errors can be tricky because it requires you to work backward by looking at the output of the program and trying to figure out what it is doing.
  - Good programing style helps you caught the semantic bugs

# Experimental debugging

- One of the most important skills you will acquire is **debugging**. Although it can be frustrating, debugging is one of the most intellectually rich, challenging, and interesting parts of programming.
- In some ways, debugging is like detective work. You are confronted with clues, and you have to infer the processes and events that led to the results you see.

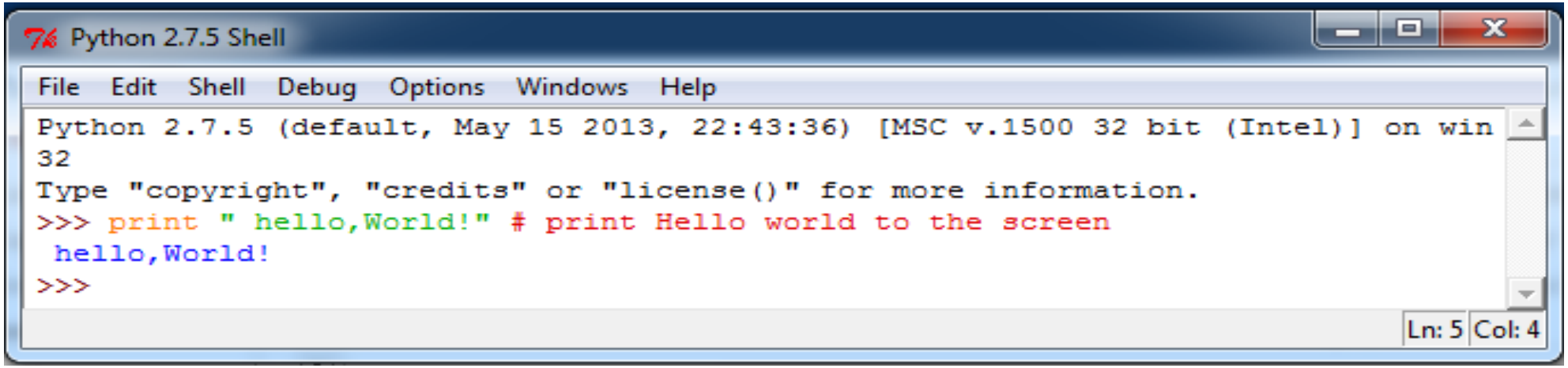
# Start with Python!
# Python IDLE

- **I**ntegrated **D**eve**l**opment **E**nvironment
- It has a Python shell window, which gives you access to the Python interactive mode. *(interactive Mode)*
- It also has a file editor that lets you create and edit existing Python source files. *(Script Mode)*



# Python IDLE

- **In Command-line Mode**
  - type Python programs and the interpreter prints the result




  - The first line of this example is the command that starts the Python interpreter.
  - The next two lines are messages from the interpreter
  - The third line starts with >>>, prompt the interpreter uses to indicate it is **ready**

# Python IDLE

- You can type Python code directly into this shell, at the '>>>' prompt. Whenever you enter a complete code fragment, it will be executed.
- Example:
  - `print (x)` : Prints the value of the expression x, followed by a newline.
    - Type `print("Hello World")` and press ENTER

```
Python 3.0.1 (r301:69561, Feb 1
win32
Type "copyright", "credits" or
>>> print ("Hello World")
Hello World
>>>
```

# Python IDLE

- IDLE can also be used as a calculator:

```
>>> 8+6
14
>>>
```

- Addition, subtraction, multiplication operators are built into the Python language. If you want to use a square root in your calculation you need to *import* the *math* module.

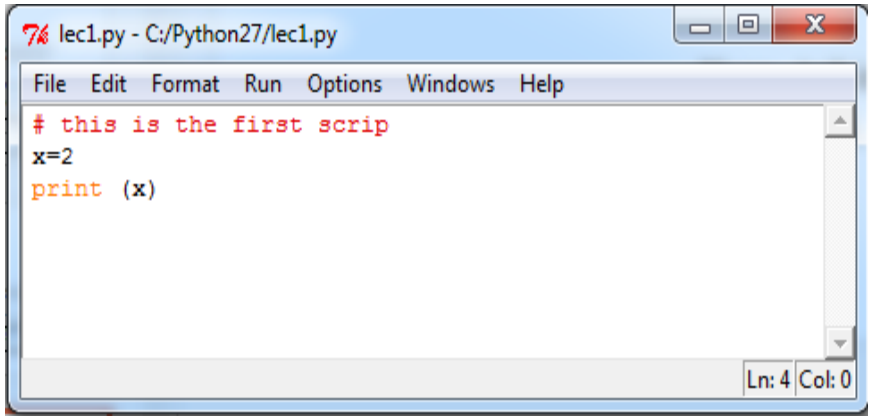
  (Do not worry about what it means right now, we will cover this later during the course)

```
>>> import math
>>> math.sqrt(67)
8.1853527718724504
>>>
```

```
>>> print ("cos(3) : ",  math.cos(3))
cos(3) :  -0.9899924966
>>> print ("cos(0) : ",  math.cos(0))
cos(0) :  1.0
>>>
```

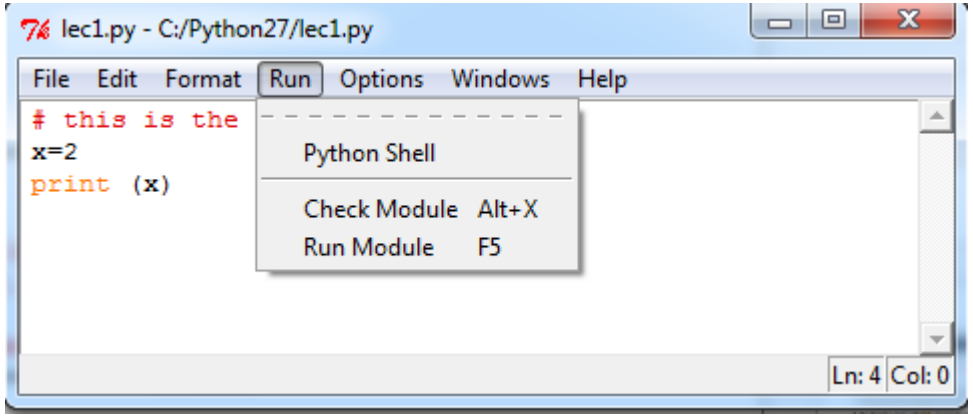
# The Python Programming Language

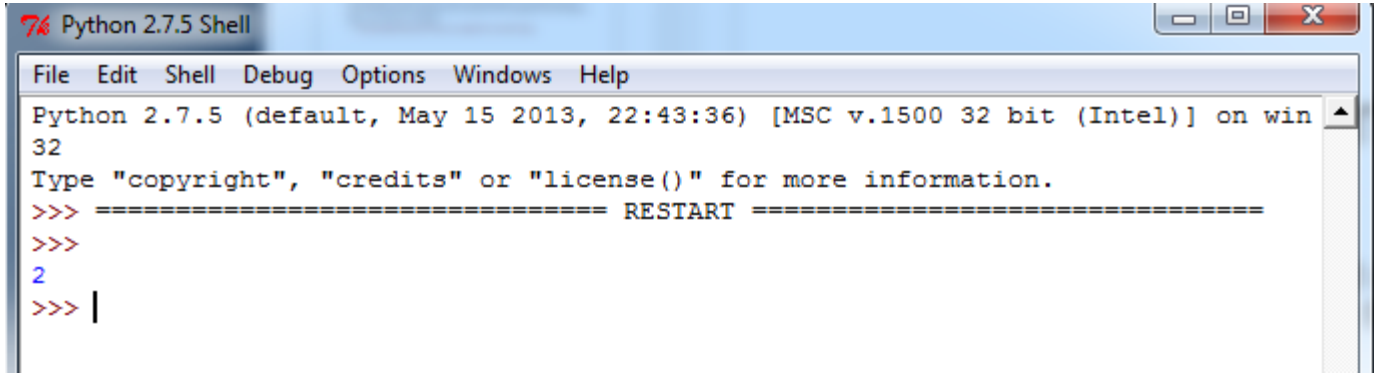
- **In Script Mode**
  - write a program in a file and use the interpreter to execute the contents of the file. The file is called a script
  - file name ends with ".py"




Then tell interpreter the name of the script by

- press F5.



# First Program: “Hello World”

in the Python

```
print "Hello, World!"
```

in the C

```
#include <stdio.h>
int main(void) {
   printf("hello, world\n");
   return 0;
}
```

in the C++

```
#include <iostream.h>
void main(){
    cout << "Hello, world." << endl;
}
```

in the Java

```
public class HelloWorld{
  public static void main(String args[]){
    System.out.println(“Hello, World!”);
  }
}
```

# Variables

- In Python, variables are designed to hold specific types of information.
- These values are belongs to different **types**
- Types in Python
  - Boolean
    - Variables of this type can be either True or False.
  - Integer
    - An integer is a number without a fractional part,
    - e.g. -4, 5, 0, -3.
  - Float
    - Any rational number, e.g. 3.432.
  - String
    - Any sequence of characters.
    - E.g. “Hello World

# Variables

- Remember, variables are containers for storing information
- Example:

```
a = "Hello World!"
print (a)
```

output

```
>>>
Hello World!
```

- The = sign is an assignment operator which says: assign the value “Hello World!” to the variable a.

```
a = "Hello World!"
a = "COMPE 111 People"
print (a)
```

output

```
>>>
COMPE 111 People
```

# Exercise 1

- Write a program that stores the value 5 in a variable a and prints out the value of a, then stores the value 7 in a and prints out the value of a (4 lines.)
- Output:

```
>>>
5
7
```

- Answer:

```
a = 5
print (a)
a = 7
print (a)
```

# Exercise 2

- What is the output of the following code?

```
a=5
b=a+7
a = 10
print (b)
```

# Exercise 2

- What is the output of the following code?

```
a=5
b=a+7
a = 10
print (b)
```

- Answer: 12

# Exercise

- Calculate
  - 23.0 to the 5th power
  - Positive root of the following equation:
    - 34*x^2 + 68*x – 510
    - Recall:
      - a*x^2 + b*x + c
      - x1 = ( - b + sqrt ( b*b - 4*a*c ) ) / ( 2*a)

# Exercise

- Answer:
  - 23.0 to the 5th power

```
23** 5     OR        import math
                     math.pow(23.0,5)
 6436343             6436343.0
```

  - Positive root of the following equation:

```
import math
a=34
b=68
c=-510
x = (-b + math.sqrt(b*b-4*a*c))/ (2*a)
print (x)
```

# Values and Types

- If you are not sure what type a value has, the interpreter can tell you:

```
>>> type('Hello, World!')
<type 'str'>
>>> type(17)
<type 'int'>
>>> type(3.2)
<type 'float'>
```

- `str`->String, `int`->Integer, `float`->floating-point

# Exercise

- What about values like ’17’ and ’3.2’?

```
>>> type(’17’)
<type ’str’>
>>> type(’3.2’)
<type ’str’>
```

They’re strings.

# Variable names and keywords

- choose meaningful names
- both letters and numbers, but begin with a letter
- `Message` and `message` are different (use lowercase by convention)
- use underscore character (_) in names with multiple words
  - person_name

# Examples

- If you give a variable an illegal name, you get a syntax error:

```
>>> 76tables = ‘seventy six tables’
SyntaxError: invalid syntax
>>> more$ = 1000000
SyntaxError: invalid syntax
>>> class = ’COMPE 111’
SyntaxError: invalid syntax
```

# Variable names and **keywords**

- Keywords define the language's rules and structure
- Keywords cannot be used as variable names
- Python has twenty-nine keywords:

```
and       def       exec      if        not       return
assert    del       finally   import    or        try
break     elif      for       in        pass      while
class     else      from      is        print     yield
continue  except    global    lambda    raise
```

# Operators

- Addition, Subtraction, Multiplication, Division: a+b, a-b, a*b, a/b respectively.
- Modulo: a % b
- Exponentiation ($a^b$): a ** b.
- Concatenation: a+b.
  - Combines two strings into one.
  - Example: “Hel” + “lo” would yield “Hello”
- * operator also works on strings; it performs repetition.
  - `'Fun'*3` is `'FunFunFun'`

# Order of Operations

- The order of evaluation depends on the rules of precedence.
- Use acronym **PEMDAS** to remember order of precedence:
  - Parentheses have the highest precedence
    - 2 * (3-1) is 4, and (1+1)**(5-2) is 8
  - Exponentiation has the next highest precedence,
    - 2**1+1 is 3 and not 4, and 3*1**3 is 3 and not 27
  - Multiplication and Division have the same precedence, which is higher than Addition and Subtraction
    - 2*3-1yields 5 rather than 4, and 2/3-1 is -1, not 1
  - Operators with the same precedence are evaluated from left to right.
    - 6*100/60 yields 10.0

# Example

- What would be the output of the followind code?

```
a = (3+4+21) / 7
b = (9*4) / (2+1) -6
print ((a*b)-(a+b))
```

# Example

- What would be the output of the followind code?

```
a = (3+4+21) / 7
b = (9*4) / (2+1) -6
print ((a*b)-(a+b))
```

- Answer: 14

# Exercise

- What would be the output of the following code?

```
print (13 + 6)
print (2 ** 3)
print (2 * (1 + 3))
print (18 / 9)
print ("13" + "6")
```

# Exercise

- What would be the output of the following code?

```
print (13 + 6)
print (2 ** 3)
print (2 * (1 + 3))
print (18 / 9)
print ("13" + "6")
```

- Answer: 14

```
19
8
8
2.0
136
```

# Warning!

- There are limits on where you can use certain expressions.
- For example, the left-hand side of an assignment statement has to be a variable name, not an expression.
- The following is illegal:

```
minute+1 = hour
```

# Comments

- Notes to your programs to explain in natural language what the program is doing, called **comments**, and they are marked with the **#** symbol
- Everything from the # to the end of the line is ignored—it has no effect on the program

```
# compute the percentage of the hour that has elapsed
percentage = (minute*100)/60   # caution:integer division
```

# Input from User

- Raw_input(): reads a string of data
  - `name = raw_input(“Enter your name:”)`

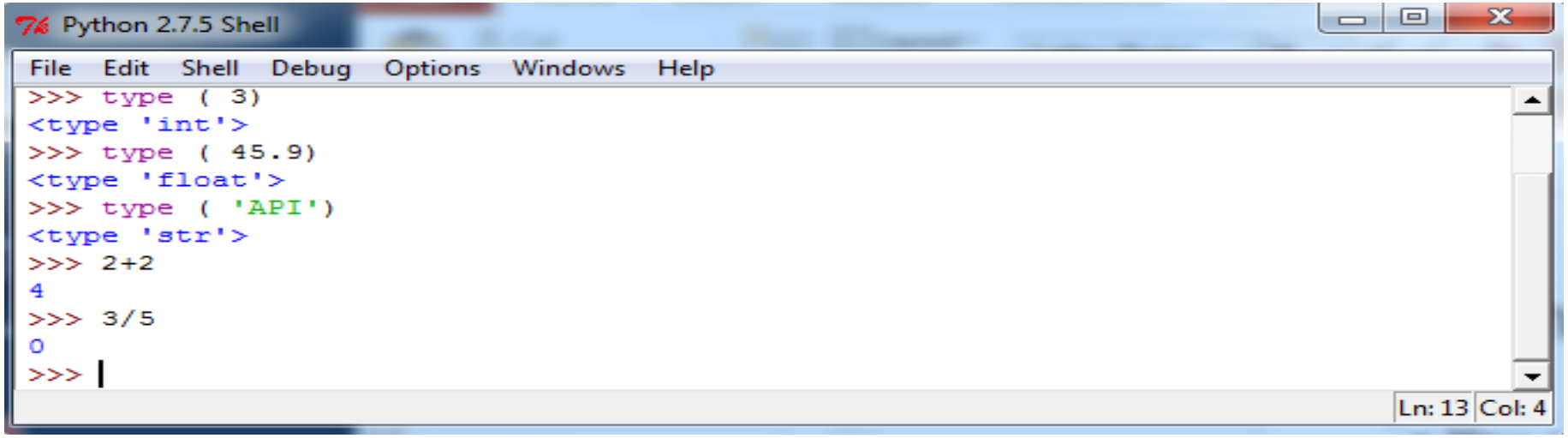
# Exercise

- Write a program that does the following in order:
  1. Asks the user to enter his/her last name.
  2. Asks the user to enter his/her first name.
  3. Prints out the user’s first and last names in that order.

To create any kind of expression we need

1. Values : A **value** is one of the basic things a program works with, like a letter( string) or a number ( int , float) .

what is the type of ‘ 53’ ?



- In python expression is operand, operator, operand. I give it to the interpreter and it gives me back the value!!

2. Operation: +,-,*,/,**

3. variables: A variable is a name that refers to a value stored.

Assignment statement creates new variables and gives them values

-MyString = ‘ Python programing’

-X= 32

-Pi= 3.14

# More!

Python 2.7.5 Shell

File Edit Shell Debug Options Windows Help

```
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> print 52*7
364
>>> print ' abc'
 abc
>>> print "abc"
abc
>>> print 'abc"
SyntaxError: EOL while scanning string literal
>>> print '3' *3
333
>>> print 5* 'a'
aaaaa
>>> print '52a'
52a
>>> 52a
SyntaxError: invalid syntax
>>> 3**5
243
>>> 3/5
0
>>> 3.0/5
0.6
>>> print 'abc' + 'de'
abcde
>>>
```

Ln: 27 Col: 4

Next Time

Control Structures!