
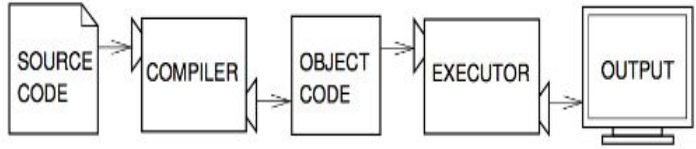


<p>Lecture #1</p>	<p>Chapter #1: The way of the program:</p> <p>The single most important skill for a computer scientist is problem solving.</p> <p>Problem solving: means the ability to formulate problems, think creatively about solutions, and express a solution clearly and accurately.</p> <p>Python is an example of a high-level language; other high-level languages you might have heard of are C, C++, Perl, and Java.</p> <p>Two kinds of programs process high-level languages into low-level languages: interpreters and compilers.</p> <p>Interpreter:</p> <p>An interpreter reads a high-level program and executes it, meaning that it does what the program says.</p> <p>It processes the program a little at a time, alternately reading lines and performing Computations</p>  <pre>graph LR; SC[SOURCE CODE] --> I[INTERPRETER]; I --> O[OUTPUT]</pre> <p>COMPLIERS:</p> <p>A compiler reads the program and translates it completely before the program starts running.</p>  <pre>graph LR; SC[SOURCE CODE] --> C[COMPILER]; C --> OC[OBJECT CODE]; OC --> E[EXECUTOR]; E --> O[OUTPUT]</pre> <p>PYTHON3 uses an INTERPRETER</p>
-------------------	--

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.

Input: Get data from the keyboard, a file, or some other device.

Output: Display data on the screen or send data to a file or other device.

Math: Perform basic mathematical operations like addition and multiplication.

Conditional Execution: Check for certain conditions and execute the appropriate sequence of statements.

Repetition: Perform some action repeatedly, usually with some variation.

For whimsical reasons, programming errors are called **bugs** and the process of tracking them down and correcting them is called **debugging**.

Syntax Errors - refers to the structure of a program and the rules about that structure. (Grammar)

- For example, in English, a sentence must begin with a capital letter and end with a period. this sentence contains a syntax error.

Runtime Errors - the error does not appear until you run the program. These errors are also called exceptions because they usually indicate that something exceptional (and bad) has happened. (Passes syntax error (there aren't any error with the structure) but there are some mistakes with logic no exceptions thrown for invalid input for ex.)

Semantic Errors - If there is a semantic error in your

program, it will run successfully, in the sense that the computer will not generate any error messages, but it will not do the right thing. It will do something else. Specifically, it will do what you told it to do. (This is when it passes syntax but the logic is completely wrong)

Natural languages: are the languages that people speak, such as English, Spanish, and French.

They were not designed by people, they evolved naturally.

Formal languages: Formal languages are languages that are designed by people for specific applications.

All of the programming languages are formal languages that have been designed to express computations.

Tokens are the basic elements of the language, such as words, numbers, and chemical elements.

When you read a sentence in English or a statement in a formal language, you have to figure out what the structure of the sentence is (although in a natural language you do this subconsciously).

This process is called **parsing**.

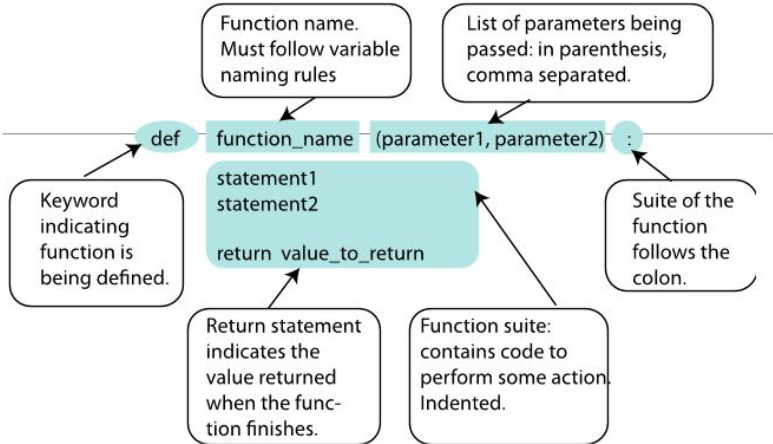
Differences between formal and natural languages:

ambiguity: Natural languages are full of ambiguity, which people deal with by using contextual clues and other information. Formal languages are designed to be nearly or completely unambiguous, which means that any statement has exactly one meaning, regardless of context.

redundancy: In order to make up for ambiguity and reduce misunderstandings, natural languages employ lots of redundancy. As a result, they are often verbose. Formal languages are less redundant and more concise.

literalness: Natural languages are full of idiom and metaphor. If I say, "The other shoe fell," there is probably no shoe and nothing falling. Formal languages mean exactly

	<p>what they say.</p> <p>Note:</p> <p>\n → nextline</p> <p>\t → tab (leave space b/w sentences)</p>
Lecture #2	<p>A program is a sequence of instructions.</p> <p>Python is an <i>interpreted</i> language:</p> <p>Interpreted means that Python looks at each instruction, <i>one at a time</i>, and turns that instruction into something that can be run.</p> <p>The = sign is the assignment statement</p> <p>A module is just a file of python commands</p> <p>Statements are commands in Python:</p> <p>They perform some action, often called a side effect, but they <i>do not return any values</i></p> <p>Expressions perform some operation and <i>return a value</i></p> <p>Whitespace are characters that don't print (blanks, tabs, etc.)</p> <p>Python Tokens:</p> <p>And del from not while as elif global or with dir else if Pass yield break except import print class exec in raise Continue finally is return def for lambda try</p> <p>A variable is a name we designate to represent an object (number, data structure, function, etc.)</p> <p>A namespace is the table that contains the association of a name with a value</p> <p>integers: 5</p> <p>floats: 1.2</p> <p>booleans: True, False</p> <p>strings: "anything" or 'something'</p>

	<p>lists: <code>[,] ['a',1,1.3]</code></p>
Lecture #3	<p>From Mathematics we know that functions perform some operation and <i>return one value</i>.</p> <p>The return statement indicates the value that is returned by the function</p> <p>This statement is optional (the function can return nothing). If no return, function is often called a procedure.</p> <p>Recursive functions are powerful mechanism in programming.</p> <p>Unfortunately, they are not always effective and often lead to mistakes.</p> <p>The most common error is infinite recursion, when the chain of function calls never ends (well, actually, it ends when you run out of free memory in your computer).</p> <p>The lambda operator or lambda function is a way to create small anonymous functions.</p> <p>Structure of a function:</p>  <pre> def function_name (parameter1, parameter2) : statement1 statement2 return value_to_return </pre> <p>Recursion example:</p>

```

1 def factorial(n):
2     if n == 0:
3         return 1
4     else:
5         return n * factorial(n - 1)
6
7 print(factorial(5))

```

Arguments:

Python Invocation

Math: $F = \text{celsius_to_Fahrenheit}(C)$

Python, the invocation is much the same

```
F = celsius_to_Fahrenheit(cel_float)
```

Terminology: `cel_float` is the *argument*

Parameters:

Function definition

Math: $g(C) = C * 1.8 + 32.0$

Python

```
def celsius_to_Fahrenheit(param_float):
    return param_float * 1.8 + 32.0
```

Terminology: `param_float` is the *parameter*

Lecture 4

Control structures allow us to alter this sequential program flow.

decision structures are statements that allow a program to execute different sequences of instructions for different cases, allowing the program to “choose” an appropriate course of action.

One Way decisions

The Python **if statement** is used to implement the decision.

```
if <condition>:  
<body>
```

The body is a sequence of one or more statements indented under the if heading.

The semantics of the if should be clear.

First, the condition in the heading is evaluated.

If the condition is true, the sequence of statements in the body is executed, and then control passes to the next statement in the program.

If the condition is false, the statements in the body are skipped, and control passes to the next statement in the program.

The body of the if either executes or not depending on the condition. In any case, control then passes to the next statement after the if.

This is a *one-way* or *simple* decision.

Relation operators

Python	Mathematics	Meaning
<	<	Less than
<=	≤	Less than or equal to
==	=	Equal to
>=	≥	Greater than or equal to
>	>	Greater than
!=	≠	Not equal to

Conditions may compare either numbers or strings.

When comparing strings, the ordering is lexicographic, meaning that the strings are sorted based on the underlying Unicode.

Because of this, all upper-case Latin letters come before lower-case letters. (“Bbbb” comes before “aaaa”)

Because of this, all upper-case Latin letters come before lower-case letters. (“Bbbb” comes before “aaaa”)

Conditions are based on Boolean expressions, named for the English mathematician George Boole.

When a Boolean expression is evaluated, it produces either a value of true (meaning the condition holds), or it produces false (it does not hold).

Some computer languages use 1 and 0 to represent “true” and “false”.

Two Way Decision

In Python, a two-way decision can be implemented by attaching an else clause onto an if clause.

This is called an if-else statement:

```
if <condition>:  
    <statements>  
else:  
    <statements>
```

When Python encounters this structure, it first evaluates the condition. If the condition is true, the statements under the if are executed.

If the condition is false, the statements under the else are executed.

In either case, the statements following the if-else are executed after either set of statements are executed.

Multi Way Decision

imagine if we needed to make a five-way decision using nesting. The if-else statements would be nested four levels deep!

There is a construct in Python that achieves this, combining an else followed immediately by an if into a single elif.


```
if <condition1>:  
    <case1 statements>  
elif <condition2>:  
    <case2 statements>  
elif <condition3>:  
    <case3 statements>  
...  
else:  
    <default statements>
```

This form sets off any number of mutually exclusive code blocks.

Python evaluates each condition in turn looking for the first one that is true. If a true condition is found, the statements indented under that condition are executed, and control passes to the next statement after the entire if-elif-else.

If none are true, the statements under else are performed.

Exception Handling

The programmer can write code that catches and deals with errors that arise while the program is running, i.e., “Do these steps, and if any problem crops up, handle it this way.”

This approach obviates the need to do explicit checking at each step in the algorithm.

The try statement has the following form:

```
try:  
<body>  
except <ErrorType>:  
<handler>
```

When Python encounters a try statement, it attempts to execute the statements inside the body.

If there is no error, control passes to the next statement after the try...except. If an error occurs while executing the body, Python looks for an except clause with a matching error type. If one is found, the handler code is executed.

The multiple excepts act like elifs. If an error occurs, Python

	<p>will try each except looking for one that matches the type of error.</p> <p>The bare except at the bottom acts like an else and catches any errors without a specific match.</p> <p>If there was no bare except at the end and none of the except clauses match, the program would still crash and report an error.</p>
Lecture #5:	<p>For Loops:</p> <h2>Definite Loops</h2> <ul style="list-style-type: none"> • A <i>definite</i> loop executes a definite number of times, i.e., at the time Python starts the loop it knows exactly how many <i>iterations</i> to do. <p>The <code>for</code> loop is a definite loop, meaning that the number of iterations is determined when the loop starts.</p> <p>While Loops:</p> <p>The <i>indefinite</i> or <i>conditional</i> loop keeps iterating until certain conditions are met.</p> <pre>while <condition>: <body></pre> <p><code>condition</code> is a Boolean expression, just like in <code>if</code> statements. The body is a sequence of one or more statements.</p> <p>Semantically, the body of the loop executes repeatedly as long as the condition remains true. When the condition is false, the loop terminates.</p> <p>Iterative Loops (Are interactive For Loops)</p> <p>One good use of the indefinite loop is to write <i>interactive loops</i>. Interactive loops allow a user to repeat certain portions of a program on demand.</p>

	<p>Sentinel Loops (Are While Loops)</p> <p>A <i>sentinel loop</i> continues to process data until reaching a special value that signals the end.</p> <p>This special value is called the <i>sentinel</i>.</p> <p>Boolean Operations: True False As well as: <i>and</i>, <i>or</i>, <i>not</i></p> <h2>Boolean Operators</h2> <hr/> <p>The Boolean operators <code>and</code> and <code>or</code> are used to combine two Boolean expressions and produce a Boolean result.</p> <pre><expr> and <expr> <expr> or <expr></pre> <p>Look at the slide if you want to see code on how to implement each.</p>
--	---

More info:

Random HI

<https://www.pythonforbeginners.com/random/how-to-use-the-random-module-in-python>

All the functions within randint in python:

<https://docs.python.org/2/library/random.html>