

Language Theory

Language type	Explanation	Example Language	Example Code
Functional	<ul style="list-style-type: none"> - Objects are constant or a function. - Order of operation is unspecified. - There is no state. 	<ul style="list-style-type: none"> - Prolog - ML - Haskell 	
Procedural	<ul style="list-style-type: none"> - A program is a state machine. - Variables change value. 	<ul style="list-style-type: none"> - C - Java - Perl 	
Imperative	<ul style="list-style-type: none"> - Explicit progress (i.e. statements) - Implicit goals 	<ul style="list-style-type: none"> - C++ - Smalltalk 	
Declarative	<ul style="list-style-type: none"> - Explicit goals - Implicit progress 	<ul style="list-style-type: none"> - Ruby on Rails - XSLT 	
Dynamic Typing	<ul style="list-style-type: none"> - Variables can hold different types of data. - The compiler knows the exact data type. 	<ul style="list-style-type: none"> - Python 	<pre>c = 1 c = "a string" c = [a, b, c]</pre>
Static Typing	<ul style="list-style-type: none"> - Variables can hold only one kind of data. - The run-time system must guess what type is being used. 	<ul style="list-style-type: none"> - C 	<pre>double c; c = 5.2; c = "a string"</pre>
Weak Typing	<ul style="list-style-type: none"> - Variables are interpreted as having multiple types. - The language automatically changes types to make an operation succeed. 	<ul style="list-style-type: none"> - Visual Basic 	<pre>var x = 5; var y = "hi" x + y // 5hi</pre>
Strong Typing	<ul style="list-style-type: none"> - Variables have only one type. - The language does not allow an operation to succeed if the types are wrong. 	<ul style="list-style-type: none"> - Python 	<pre>v = '1.2' x = 5 * b #runtime error</pre>
Safe	<ul style="list-style-type: none"> - The language does not allow operations which can cause an error. 	<ul style="list-style-type: none"> - Visual Basic 	<pre>var x = 5; var y = "hi" x + y // 5hi</pre>
Unsafe	<ul style="list-style-type: none"> - The language allows operations which can cause an error. 	<ul style="list-style-type: none"> - C 	<pre>int x = 5; char y[] = "hi"; char* z = x + y;</pre>
Nominative Typing	<ul style="list-style-type: none"> - Variables are comparable only if they have the same type. 	<ul style="list-style-type: none"> - Java 	<pre>public int add (int a, in b) { return (a + b); }</pre>
Structural Typing	<ul style="list-style-type: none"> - Variables are comparable if they have the same structure. 	<ul style="list-style-type: none"> - Haskell 	<pre>add a b = a + b</pre>
Duck Typing	<ul style="list-style-type: none"> - An object's current set of methods and properties determines the valid semantics, rather than its inheritance from a particular class. 	<ul style="list-style-type: none"> - Python 	
Explicit Memory Management	<ul style="list-style-type: none"> - The program has to manage the memory. 	<ul style="list-style-type: none"> - C - C++ 	<pre>char* str = malloc(1024); free(str);</pre>
Implicit Memory Management	<ul style="list-style-type: none"> - The memory management is done automatically by the language for the programmer. 	<ul style="list-style-type: none"> - Python - Perl - PHP - Matlab - sh 	

Regular expressions

Meta characters

Character	Meaning	Example (given <code>The Theatre is not open</code>)
<code>[]</code>	Matches any of the characters in the brackets	<code>e[a][tT]</code> matches <code>e T</code> and <code>eat</code> <code>[a-d]</code> matches any of the characters <code>a,b,c,d</code>
<code> </code>	Matches one of the items of either side	<code>t(he or)e</code> matches <code>thee</code> or <code>tore</code>
<code>.</code>	Matches any single character	<code>t.</code> matches <code>te</code> but not <code>t</code>
<code>{}</code>	Allows quantifying	- <code>{n,m}</code> matches at least <code>n</code> and at most <code>m</code> of the preceding item - <code>{n}</code> matches exactly <code>n</code> of the preceding item - <code>{,m}</code> matches at most <code>m</code> of the preceding item - <code>{n,}</code> matches at least <code>n</code> of the preceding item
<code>?</code>	Matches zero or one of the preceding item	<code>Th?h</code> matched the first <code>Th</code>
<code>+</code>	Matches one or more of the preceding item	<code>Th+e</code> matches the first <code>The</code>
<code>*</code>	Matches any numbers of the preceding item	<code>T.*h</code> matches <code>The Th</code>
<code>()</code>	Groups the items matched in the parentheses	<code>Th(eatr)?e</code> matches the first <code>The</code>
<code>^</code>	Matches the beginning of the line OR symbolizes negation inside brackets	- <code>^Th</code> matches the first <code>Th</code> but not the second - <code>[^0-9]</code> matches any character which is not a digit
<code>\$</code>	Matches the end of a line	<code>n\$</code> matches the last <code>n</code>
<code>\b</code>	Match at a word boundary	<code>The\b.</code> matches <code>"The "</code>
<code>\B</code>	Match except at a word boundary	<code>The\B.</code> matches <code>Thea</code> but not <code>"The "</code>
<code>\d</code>	A digit <code>[0-9]</code>	
<code>\D</code>	A non-digit <code>[^0-9]</code>	
<code>\w</code>	A word character <code>[a-zA-Z_0-9]</code>	
<code>\W</code>	A non-word character <code>[^a-zA-Z_0-9]</code>	
<code>\s</code>	A white space character <code>[\t\n\r\f]</code>	
<code>\S</code>	A non-white space character <code>[^\t\n\r\f]</code>	

Regular Expressions in Perl

- `$string =~ m/regex/; #returns 1` iff the LHS matches the RHS regular expression
- `$string !~ m/regex/; #returns 1` iff the LHS doesn't match the RHS regular expression
- `$string =~ s/source/dest/; #replaces the first occurrence of source with dest`
- `$string =~ s/source/dest/g; #replaces all occurrences of source with dest`