



The Pragmatic Language

Language Primer

This document is designed to help progammers learn the basics of Bill.

Copyright (c) 2022 J. Smith. Permission is granted to copy, distribute and/or modify this document, provided this notice is kept.

1 Introduction	1
1.1 Reserved Words	2
2 Types	3
2.1 Integers	3
2.1.1 Unsigned	4
2.2 Floats	4
2.3 Boolean	4
2.4 Strings	4
2.5 Null	4
2.6 Others	4
3 Expressions	5
3.1 Math	5
3.2 Concatenation	5
4 Containers	7
4.1 Arrays	7
4.2 Vectors	7
4.3 Sets	7
4.4 Tuples	7
4.5 Dictionaries	7
4.6 Others	7
5 Flow Control	9
5.1 Conditions	9
5.2 If	9
5.3 Else	9
5.4 Elsif	9
5.5 While	9
5.6 For	10
5.7 Traversals	10
6 Functions	11
6.1 Calling a Function	11
6.2 Declaring a Function	11
6.3 Return	11
6.4 Used in Expressions	11
6.5 Built-in	11
7 Exceptions	13
7.1 Throw	13
7.2 Try	13
7.3 Catch	13

8 Style	15
8.1 Comments	15
8.2 Statements	15
8.3 Blocks	15

Chapter 1

Introduction

Name: `bill` *Beginner – Intermediate Learning Language*

Purpose: It is a general purpose statically typed, with dynamic types, easy to use language.

Key uses:

- Learning basic level programming.
- Pursuing enthusiast programming.

Features:

- portable
- a variety of useful data types
- extend-able data types
- simple to use
- constants
- ease of static typing
- strong static typing

Hello World

"Hello World" is traditionally the first program one writes in a new language. That makes it a good starting point.

```
#!/usr/bin/env bill

# hello.bill
# aka hello world

fun main():no_value
{
    writeln("Hello World")
    exit // defaults to 0
}
```

hello.bill

Of course, the first few lines are unnecessary. However, declaring a main function is required. For details, on that, in chapter 6 ([Functions](#)).

To compile and run:

```
prompt> bill hello.bill
```

If your operating system supports shebangs, the following will work:

```
prompt> hello.bill
```

1.1 Reserved Words

Here is a list of Reserved Words:

break	exit loop*
continue	skip to next iteration*
else	default condition*
elsif	subsequent condition*
exit	end program (possible exit value)
if	condition*
return	end a function (possible return value)**
try	begin try block***

With exit, you can use any positive int64 value. However, 0 (default) generally represents a good run. Typically, a problem is designated by 1. However other values are possible.

* See [Flow Control](#).

** See [Functions](#).

*** See [Exceptions](#).

Next: [Types](#)

Chapter 2

Types

Types

Types

Every form of data is a type. By extension, the same is true for functions. Built in types:

- null
- boolean
- int8
- int64
- float64
- string
- array
- vector
- set
- tuple
- dictionary

2.1 Integers

All integers are signed. See [Unsigned](#) for more information.

There are two integer types:

- int8 is 8 bit. (-128 to + 127)
- int64 is 64 bit. (\pm a really big number)

Tip: Only use int8 for space conserving situations, provided all values will always fall within limits. Else, use int64. Don't trust "There's no reason for it to go beyond limits." It must be **impossible** to exceed limits or, it's only a matter of time.

2.1.1 Unsigned

Why no unsigned integers? Here is a great answer: <https://blog.robertelder.org/signed-or-unsigned-part>

2.2 Floats

2.3 Boolean

2.4 Strings

2.5 Null

2.6 Others

For more information, please see [Containers](#).

Next: [Expressions](#)

Chapter 3

Expressions

[Expressions](#)

Express!

3.1 Math

3.2 Concatenation

Next: [Containers](#)

Chapter 4

Containers

[Containers](#)

Containers can hold multiple values.

4.1 Arrays

4.2 Vectors

4.3 Sets

4.4 Tuples

4.5 Dictionaries

4.6 Others

Next: [Flow Control](#) Flow Control

Chapter 5

Flow Control

Flow Control

If...

5.1 Conditions

5.2 If

5.3 Else

5.4 Elsif

5.5 While

```
# whileloop.bill
# while loop syntax sample

fun main(argv):int8
{
    while true:
    {
        # This is the loop that never ends.
    }

    while 1 > 3:
    {
```

```
        # This loop is skipped.
    }

//    var $limit:vector    = $argv[1]
var i:int8 = 0
while i < argv[1]:
{
    writeln(i)
    i ++
}
}
```

5.6 For

```
# forloop.bill
# for loop syntax sample

/* @fn      main
 * @brief   simple forloop
 */
fun main():no_value
{
    for var i:int64 = 0 to 10:
    {
        writeln(i)
    }

    for (var i:int64 = 0; i < 1):
    {
        writeln(i)
    }

    for (var i:float64 = 0; i < 1; i += .03):
    {
        writeln(i)
    }

    # foreach (sequence types only)
    //    foreach(var str arg, argv):
    //    {
    //        writeln(arg);
    //    }
}
```

5.7 Traversals

Next: [Functions](#)

Chapter 6

Functions

Functions

Encapsulating repeatable steps, is how we make programming easier.

6.1 Calling a Function

6.2 Declaring a Function

As seen in the Introduction, we have simple declarations.

```
#!/usr/bin/env bill

# hello.bill
# aka hello world

fun main():no_value
{
    writeln("Hello World")
    exit // defaults to 0
}
```

hello.bill

The function definition line should look familiar. The "fun" reserved word declares a function.

After the ":" is the function return type. However, in this case "no_value" indicates there is no return of any kind.

A common practice of statically typed languages is to declare the type "void," which is similar.

6.3 Return

6.4 Used in Expressions

6.5 Built-in

Here is a list of built-in functions:

<code>catch()</code>	catch exception
<code>throw()</code>	throw exception
<code>toFloat()</code>	convert to float
<code>toInt()</code>	convert to integer
<code>toString()</code>	convert to string
<code>toTuple()</code>	convert to tuple
<code>type()</code>	get an object's type
<code>write()</code>	print (without newline)
<code>writeln()</code>	print (with newline)

Next: [Exceptions](#)

Chapter 7

Exceptions

[Exceptions](#)

oops!

7.1 Throw

7.2 Try

7.3 Catch

Next: [Style](#)

Chapter 8

Style

Style

Code style can be a matter of choice...

However, consistency means readability. As such, here are conventions used throughout this documentation.

8.1 Comments

Possible comment types:

```
# This is a comment type recommended for shebangs.
// This is the recommended end-of-line comment.
/* This type of comment is recommended for documentation blocks. */

// or

/* myfunction
 * Demo an operation
 */
```

8.2 Statements

8.3 Blocks

```
Declaration:    // if, while, etc...
{
    // Code here.
}
```

