9. The Language Specification - Arithmetic

This is the first glimpse in the specification you are getting of multi-use functions. They will be organized as subsections under each section. Each subsection will be labelled by the type of arguments it receives. Here are the possible types of arguments:

```
Integer: An integer
Number: An integer or floating point decimal
String: A string of characters
List: A list of arbitrary elements
Set: A Python Set
Sequence: A list, tuple, or string
Collection: A list, tuple, string or set
Any: Anything
```

9.1. “+” - Addition

Arity: 2

9.1.1. Num a, Num b: Addition

This simply returns the sum of the two numbers.

Ex:

```
==================================================
+2 2
==================================================
Pprint("\n",plus(2,(2)))
==================================================
4
```

9.1.2. Seq a, Any b: Concatenation

If the `b` is also a sequence of the same type as `a`, it returns the concatenation of the two sequences.

Ex:

```
==================================================
+"Hello"" World!
==================================================
Pprint("\n",plus("Hello"," World!"))
==================================================
Hello World!
```

But if `b` is not a sequence of the same type as `a`, a one-element sequence of the same type as `a` containing only `b` will be concatenated to a.

Ex:

```
==================================================
+[1 2 3)4
==================================================
Pprint("\n",plus(Plist(1,(2),(3)),4))
==================================================
[1, 2, 3, 4]
```

9.2. “*” - Multiplication

Arity: 2

9.2.1. Num a, Num b: Multiplication

This returns the product of the two numbers.

Ex:

```
==================================================
*2T
==================================================
Pprint("\n",times(2,T))
==================================================
20
```

9.2.2. Seq a, Int b: Repetition

This function repeats sequence `a`, `b` times. This is the same as repetition in Python.

Ex:

```
==================================================
*"abc"5
==================================================
Pprint("\n",times("abc",5))
==================================================
abcabcabcabcabc
```

9.2.3. Seq a, Seq b: Cartesian Product

Calculates the Cartesian Product of the two sequences. They have to be of the same type. This means that it generates all the possible ways that you can select one value from both sequences.

Ex:

```
==================================================
*"abc" "123"
==================================================
Pprint("\n",times("abc",("123")))
==================================================
[('a', '1'), ('a', '2'), ('a', '3'), ('b', '1'), ('b', '2'), ('b', '3'), ('c', '1'), ('c', '2')
```

9.3. “-” - Subtraction

Arity: 2

9.3.1. Num a, Num b: Subtraction

Computes the difference of `a` from `b`.

Ex:

```
==================================================
-T4
==================================================
Pprint("\n",minus(T,4))
==================================================
6
```

9.3.2. Col a, Col b: Setwise Difference

Computes the setwise difference of `a` from `b`. This means it returns a collection with the elements in `a` that are not in `b`, using the type of `a`. It preserves the order of `a`.

Ex:

```
=================== 10 chars ====================
-[1 2 3)[2
==================================================
Pprint("\n",minus(Plist(1,(2),(3)),Plist(2)))
==================================================
[1, 3]
```

9.4. “/” - Division

Arity: 2

9.4.1. Num a, Num b: Division

Returns `a` divided by `b`. Uses integer division which means it truncates the fractional part of the answer.

Ex:

```
==================================================
/T4
==================================================
Pprint("\n",div(T,4))
==================================================
2
```

9.4.2. Seq a, any b: Count Occurrences

Returns the number of times element b appeared in sequence a.

Ex:

```
==================================================
/[1 2 3 2 5)2
==================================================
Pprint("\n",div(Plist(1,(2),(3),(2),(5)),2))
==================================================
2
```

9.5. “%” - Modulus

Arity: 2

9.5.1. Num a, Num b: Modulus

Returns the remainder when `a` is integer divided by `b`.

Ex:

```
==================================================
%T3
==================================================
Pprint("\n",mod(T,3))
==================================================
1
```

9.5.2. String a, Any b: String Formatting

This applies Python’s string formatting that normally occurs with `%`. Requires `%s` or any of the other within the string,, just like in Python.

Ex:

```
==================================================
%"a: %d"2
==================================================
Pprint("\n",mod("a: %d",2))
==================================================
a: 2
```

9.5.3. Int a, Seq b: Extended Slicing

Pyth’s slicing operator does not support extended slicing, so this operator has the effect of doing `b[::a]`. This means that it will pick every `a` elements of `b`.

Ex:

```
==================================================
%2"Hello
==================================================
Pprint("\n",mod(2,"Hello"))
==================================================
Hlo
```

9.6. “^” - Exponentiation

Arity: 2

9.6.1. Num a, Num b: Exponentiation

This raises the `a` to the power of `b`. Like Python, it allows rational exponents.

Ex:

```
==================================================
^4 2
==================================================
Pprint("\n",Ppow(4,(2)))
==================================================
16
```

9.6.2. Seq a, Int b: Cartesian Product With Repeats

Finds the Cartesian Product of `b` copies of sequence `a`. This means that it finds all possible sequences with length `b` that contain only elements from sequence `a`.

Ex:

```
==================================================
^"abc"3
==================================================
Pprint("\n",Ppow("abc",3))
==================================================
['aaa', 'aab', 'aac', 'aba', 'abb', 'abc', 'aca', 'acb', 'acc', 'baa', 'bab', 'bac', 'bba', 'bb
```

9.7. “_” - Unary Negation

Arity: 1

9.7.1. Num a: Negation

Returns the additive inverse of `a` or `-a`. There are no negative number literals in Pyth, this is how you define negatives in Pyth.

Ex:

```
==================================================
_25
==================================================
Pprint("\n",neg(25))
==================================================
-25
```

9.7.2. Seq a: Reversal

Returns `a` in reversed order. This is equivalent to the alien smiley face, `[::-1]` in Python.

Ex:

```
==================================================
_"abc"
==================================================
Pprint("\n",neg("abc"))
==================================================
cba
```

9.7.3. Dict a: Invert

Returns `a` with its keys and values swapped.

Ex:

```
=================== 7 chars =====================
XH1\a_H
==================================================
Pprint("\n",assign_at(H,1,"a"))
Pprint("\n",neg(H))
==================================================
{1: 'a'}
{'a': 1}
```

9.8. “P” - Prime Factorization

Arity: 1

9.8.1. Int a: Prime Factorization

Returns the prime factorization of `a`. Returns it as a list and multiplicities are just repeated.

Ex:

```
==================================================
P12
==================================================
Pprint("\n",primes_upper(12))
==================================================
[2, 2, 3]
```

9.8.2. Seq a: All But Last

Returns all but the last element of `a`. This is equivalent to the Python `[:-1]`

Ex:

```
==================================================
P"abc"
==================================================
Pprint("\n",primes_upper("abc"))
==================================================
ab
```