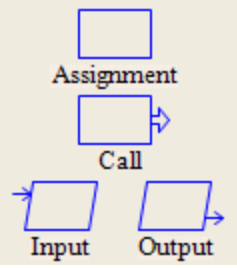
# RAPTOR Syntax and Semantics By Lt Col Schorsch

**Program** - an ordered collection of instructions that, when executed, causes the computer to behave in a predetermined manner.

**Variable** - A variable names a memory location. By using that variable's name you can store data to or retrieve data from that memory location.
A **variable** has 4 properties: ❶ a name, ❷ a memory location, ❸ a data type, ❹ a value. You can assign a value to a variable using an assignment statement (see below).
RAPTOR variables are declared on first use, they must be assigned a value on first use and based on that value it's *data type* will be Number, String, or an Array of Numbers.

**Symbols**




## Data Type - A Data Type is the name for a group of data values with similar properties.

A Data Type has 4 properties: ❶ a name, ❷ a set of values, ❸ a notation for *literals* of those values, ❹ operations and functions which can be performed on those values.

RAPTOR has two simple data types: Number and String (Array data types are described later)

| Type name | Literal Values | Operations grouped from lowest to highest precedence |
|---|---|---|
| Number | -32, 0, 1, 49, etc. -2.1, 3.1415, etc. | [=,<,<=,>,>=,/=,!=], [+, -], [*, /, rem, mod], [**,^] |
| String | "Hello", "Bob", etc. | [=,<,<=,>,>=,/=,!=], [+] |

## Operator – An operator directs the computer to perform some computation on data.

Operators are placed between the data (operands) being operated on (i.e. `X / 3`, `Y + 7`, `N < M`, etc.)

basic math operators: +, -, *, /, ^, **, rem, mod
+, -, *, / are defined as one would expect, ** and ^ are exponentiation, ex 2**4 is 16, 3^2 is 9
rem (remainder) and mod (modulus) return the remainder (what is left over) when the right operand divides the left operand, ex 10 rem 3 is 1, 10 mod 3 is 1

Concatenation operator: +
Joins strings and numbers (i.e. `"Average is " + (Total / Number))`

The following operators are only used in decisions (see Selection and Iteration)

Relational operators: =, !=, /=, <, >, >=, <=
<, >, >=, <= are defined as expected. The result of a relational comparison is a Boolean value.

Logical operators: and, or, not, xor

| Expression | Result |
|---|---|
| True and True | True |
| True and False | False |
| False and True | False |
| False and False | False |

| Expression | Result |
|---|---|
| True or True | True |
| True or False | True |
| False or True | True |
| False or False | False |

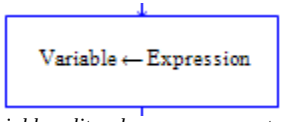
| Expression | Result |
|---|---|
| Not(True) | False |
| Not(False) | True |

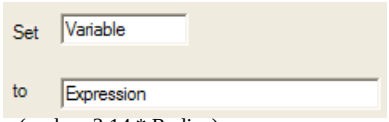
xor is true when either operand is true (but not when both operands are true).

## Assignment Statement - An assignment statement is used to evaluate an *expression* and store the results in a *variable*. The *expression* is on the right hand side of the assignment operator, ⟵.

An *expression*'s value (after it is evaluated) is stored in the *variable* on the left hand side of the ⟵ operator.
An *expression* must evaluate to a value of the same data type as the *variable* in which it is being stored.

**Syntax:**
*Variable* ⟵ *Expression*

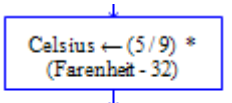


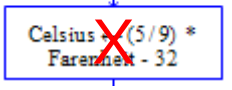

An *expression* is either a *variable*, a *literal*, or some *computation* (such as 3.14 * Radius).
A *literal* (such as 2.143, 42, "Help") evaluates to itself.
A *variable* evaluates to the data stored at its memory location.
Evaluating a *computation* involves evaluating the literals, variables, operators and functions in the expression.

| | | |
|---|---|---|
| `Age` | `← 21` | The value 21 is stored in variable Age's memory location |
| `Count` | `← Count + 1` | The value that is stored in Count's memory location is incremented by 1 |
| `Force` | `← Mass * Acc` | Mass and Acc are multiplied together, the product is stored in variable Force |
| `Delta_X` | `← abs(X2 - X1)` | Take the absolute value difference and store it in Delta_X |
| `Name` | `← "Schorsch"` | Assigns the string "Schorsch" to the variable Name's memory location |

Order of operations matters!
Precedence levels from lowest to highest
[=,<,<=,>,>=,/=,!=], [+, -], [*, /, rem, mod], [**,^]

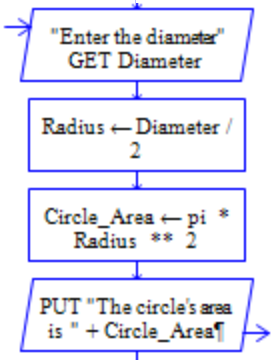


Correct Equation



Incorrect Equation

Circle Area program:
Given a diameter this program computes and displays the area of a circle with that diameter




## Function – A function performs a computation on data and returns a value.

Functions use parentheses to indicate their data (i.e. `sqrt(4.7)`, `sin(2.9)`, etc.)

Basic math: sqrt, log, abs, ceiling, floor
sqrt returns the square root, ex sqrt(4) is 2
log returns the natural logarithm, ex log(e) is 1
abs returns the absolute value, ex abs(-9) is 9
ceiling rounds up to a whole number, ex ceiling(3.14159) is 4
floor rounds down to a whole number, ex floor(10/3) is 3

Trigonometry: sin, cos, tan, cot, arcsin, arccos, arctan, arccot
Angles are in radians, ex sin(pi) is 0.
arctan and arccot are the two parameter versions of those functions. (i.e. arctan(X/Y) is written in RAPTOR as arctan(X,Y)).

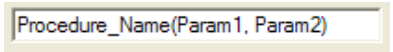
Miscellaneous: Length_Of
Length_Of returns the number of characters in a string
ex `Name ← "Stuff"` followed by `Length_Of(Name)` is 5
(also returns the number of elements in an array which you will learn later)

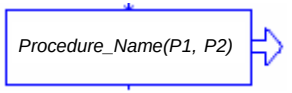
Random
Returns a random number between [0.0,1.0)
(`Random * X + Y` extends the range by X and shifts it by Y)

## Procedure Call - A procedure is a set of executable statements that have been given a name.

Calling a procedure executes the statements associated with that procedure.

*Procedure_name* (*Parameter 1, Parameter 2, etc.*)




The number and order of parameters in the call must match the expected number and order.
The data types of the parameters in the call must match the expected data types of the parameters.
Procedure parameters can be used to give (supply) a procedure with data or can accept (receive) data.
Parameters must be variables if they receive a value.
Parameters can be an expression (computation), variable or literal if they supply a value.

| | |
|---|---|
| `Delay_for(0.2)` | delays execution for 2/10ths of a second |
| `Clear_Console` | erases the master console contents |
| `Draw_Circle(X, Y, 7, Blue)` | draws a blue circle at location X,Y with a radius of 7 |