Python Guide

Terminology

Application - A program that is applied to some specific task.

Argument - Data that is input and used in a function. These go in the parentheses after a function. For example, the **`print()`** function takes a string as an argument, like **`print("Hello World")`**, where **`"Hello World"`** is the argument.

Assignment - The process of giving a variable some value.

Boolean Expression - See "Truth Statement."

Boolean Value - A value that is either **`True`** or **`False`**.

Bugs - Flaws in the code. These can be syntax errors or semantic errors.

Camelcase - A naming convention for variables where words within the name are capitalized: helloWorld, pleaseExcuseMyDearAuntSally, camelCasingIsUseful.

Conditional - Code that is only execute if some condition is true. These come in two parts: **`if`** and **`else`**. The code written in **`if`** are executed if the condition is **`True`**, and the code written in **`else`** are executed if the condition is **`False`**. These are written in Python using the **`if`** and **`else`** keyword. These are also known as if/else statements.

Constants - A variable whose value never changes once it is assigned. For example, GRAVITY = 9.8.

Counter - See "Iterator."

Decrementing - A looping technique that decreases the value of an iterator to some minimum value. When the iterator is at or below the minimum, the loop quits.

Flag - A variable that has a boolean value that signals some condition in the code.

For Loop - A type of loop the executes for some range of numbers or items. It uses a counter variable to maintain its position in the range, known as an iterator. These are written in Python using the **`for`** keyword.

Function - A separate piece of code that can be called and executed from anywhere in the main body of code. In Python, these look like **`name(argument1, argument2, argument3)`**.

if/else statement - See "Conditional."

Incrementing - A looping technique that increases the value of an iterator to some maximum. When the iterator is at or above the maximum, the loop quits.

Infinite Loop - A loop that does not end; this will stall the program forever and can crash your computer. These should always be avoided.

Input - Externally generated data that is caught by the program. This can be anything from text input by a user to sensory data obtained from a robot.

Interactive Development Environment (IDE) - A specialized text editor that is built specifically for making programs. IDLE is an IDE.

Iteration - The process in which code is repeated one or more times. Loops use iteration.

Iterator - A variable used to determine the current place in the loop, usually as a number. Typically, these are used as part of for loops. For example, in the code *for i in range(0, 10)*, the variable *i* is an iterator. This is also referred to as a counter.

Literalism - The explicit adherence to a given text. Computers interpret programs literally.

Loop - Code that is executed one or more times using iteration. This can be while some condition is true or for some range of numbers.

Nesting - Putting conditional or looping code within other conditional or looping code.

Output - Anything that the computer does to interface with the outside world. This can be text, images, lights, movement, etc. For example, the **`print`** function outputs text to the shell.

Parameter - The required input of a function. These are specified in the definition of a function. For example, in the function **`def hello(x)`**, **`x`** is a parameter.

Program - A coded behaviour that a computer can execute.

Semantics - The meaning of a piece of code. This can apply to single lines or entire programs.

Semantically Equivalent - The meaning of two pieces of code are the same. A is semantically equivalent to B if the semantics of A and B are identical.

Semantic Error - A bug that exists because the code is not interpreted by the computer as intended.

Shell - A programmer's interface that allows code to be ran line-by-line. IDLE opens to a shell.

String - A data type representing a sequence of characters, such as "Hello".

Syntax - The specific way code is written.

Syntax Error - A bug that exists because the code does not obey the syntax requirements of the programming language.

Syntax Requirements - The specific way code must be written.

Truth Statement - Code that checks for some condition and evaluates to a boolean value.

While Loop - A type of loop the executes while some condition is true. These are written in Python using the **`while`** keyword.

Variable - A representation of a number, string or other piece of data.

Syntax

Variables and Literals

- **"[value]"** - creates a string with the value. For example, *"Hello"* creates a string with the value, Hello.
- **x = v** - sets the variable, *x*, to the value, *v*. For example, *foo = 1* sets the variable, named *foo*, to the value, *1*.

Math

- **x + y** - adds the values of *x* and *y*; for strings, this put them together ("h" + "i" = "hi")
- **x - y** - subtracts the value of *x* by *y*
- **x * y** - multiplies the values of *x* and *y*
- **x / y** - divides the value of *x* by *y*
- **x % y** - gives the remainder of *x* divided by *y*
- **x ** y** - performs x^y

Truth Statements

- **x > y** - is true if x is greater than y
- **x >= y** - is true if x is greater than or equal to y
- **x < y** - is true if x is less than y
- **x <= y** - is true if x is less than or equal to y
- **x == y** - is true if x is equal to y
- **x != y** - is true if x is not equal to y
- **x in y** - is true if x is somewhere in y
- **x and y** - is true if both x and y are true
- **x or y** - is true if either x and y are true
- **not x** - is true if x is not true; x can be a truth statement in parentheses

Conditionals

- **if v:** - executes the indented code below it if the value, *v*, is true
    - **else:** - executes the indented code below it if the value, **`v`**, is false. This must be placed directly after the indented code below **`if(v)`**.

Loops

- **while v:** - executes the indented code below it as long as the value, *v*, is true

Functions

- **def name(arg1, arg2, ...)** - creates a function with the name, *name*, that takes the arguments, *arg1, arg2,* and so on.
- **return v** - returns the value, *v*, from the function it is coded within and exits the function

Predefined Functions

- **print(arg)** - prints the value of arg
- **int(arg)** - converts the argument to an integer
- **float(arg)** - converts the argument to a decimal number
- **str(arg)** - converts the argument to a string

Finch Movement Functions

- **forward()** - moves the Finch forward continuously
- **forward(inches)** - moves forward approximately the number of inches
- **forward(inches, delay)** - moves forward approximately the number of inches. If delay is **`True`**, waits for the movement to finish. If delay is **`False`**, the program continues while the movement is happening.
- **backward()** - moves the Finch backward continuously
- **backward(inches)** - moves backward approximately the number of inches
- **backward(inches, delay)** - moves backward approximately the number of inches. If delay is **`True`**, waits for the movement to finish. If delay is **`False`**, the program continues while the movement is happening.
- **turnRight()** - turns the Finch clockwise continuously
- **turnRight(angle)** - turns the Finch clockwise for approximately that angle
- **turnRight(angle, delay)** - turns the Finch clockwise to approximately that angle. If delay is **`True`**, waits for the movement to finish. If delay is **`False`**, the program continues while the movement is happening.
- **turnLeft()** - turns the Finch counterclockwise continuously
- **turnLeft(angle)** - turns the Finch counterclockwise for approximately that angle
- **turnLeft(angle, delay)** - turns the Finch counterclockwise for approximately that angle. If delay is **`True`**, waits for the movement to finish. If delay is **`False`**, the program continues while the movement is happening.
- **setWheels(left, right)** - sets the left wheel's movement to the left value and the right wheel's movement to the right value
- **stop()** - stops the Finch's wheels

Finch Output Functions

- **light(color)** - turns the Finch's nose the specified color. Color must be a string.
- **light(r, g, b)** - turns the Finch's nose to the color specified by the components. r, g, and b are the red, green, and blue color components, respectfully.
- **buzz(duration, frequency)** - makes the Finch play a sound at the specified frequency for the specified duration. The program does not wait for the buzz to finish.
- **delayedBuzz(duration, frequency)** - makes the Finch play a sound at the specified frequency for the specified duration. The program waits for the buzz to finish.

Other Finch Functions

- **halt()** - stops everything the Finch is doing