Microbric Edison Ed.Py Language Specification
Version 1.1

Brian Danilko

November 20, 2015

Contents

1 Version history **2**

2 Ed.Py language **2**

2.1 Overview . . . 2

2.2 Lexical rules . . . 2

2.3 Data types: . . . 2

2.4 Literals . . . 2

2.4.1 int literals . . . 2

2.4.2 list literals . . . 3

2.4.3 character literals (NOT sure if this will be supported!) . . . 3

2.4.4 tunestr literals . . . 3

2.5 Variables . . . 3

2.6 Operators . . . 4

2.7 Assignment . . . 4

2.8 Built in functions . . . 4

2.9 Functions . . . 4

2.10 Classes . . . 5

2.11 Control flow . . . 5

2.11.1 If statement . . . 5

2.11.2 While statement . . . 5

2.11.3 For statement . . . 5

2.12 Edison object . . . 5

2.13 Notes . . . 6

2.13.1 Compatibility with python 2/3 6
2.13.2 Import 6
2.13.3 Concurrency 6
2.13.4 OLD historical edison object notes 6

1 Version history

Ver	Date released	Notes
1.1	20/Nov/2015	Corrected pdf conversion flaws, moved old stuff to end.
1.0	19/Nov/2015	Original work plus updates from meeting of 9/Nov/2015

2 Ed.Py language

2.1 Overview

- A strict subset of python! The idea is that this strip-down python subset will introduce students to programming with a simple subset. And when they go on to full python programming, the will not have to relearn anything.

2.2 Lexical rules

- As Ed.Py is a subset of python, all of the lexical rules (comments, line continuations, indents, etc.) from python will be used for Ed.Py.
- See `https://docs.python.org/2.7/reference/lexical_analysis.html` for details on lexing
- Comment starts with '#'.

2.3 Data types:

int signed shorts (16bits)

list a list of ints with maximum number of entries.

tunestring a sequence of characters, which should be from the tune string set of valid characters (the firmware handles it if a character is NOT valid).

tunestring[x] → char. But as we don't have a char type, the programmer would have to do this:

1. int ← ord(tunestring[x]).

Similarily to assign into a tunestring, need to do this:

2. tunestring[x] = chr(int), or

With a character LITERAL, could also do this:

3. tunestring[x] = 'a'

2.4 Literals

2.4.1 int literals

(+|-)?[1-9][0-9]* a decimal int

0x[0-9a-fA-F]* a hexidecimal int (only positive)

0b[0-1]* a binary int (only positive)

True | non-zero int literal a true boolean condition

False | 0 int literal a false boolean condition

2.4.2 list literals

[A, B, . . .] a list of int literals

2.4.3 character literals (NOT sure if this will be supported!)

Note – BS means the backslash character.

'char' a single char surronded by single quotes (')

simple-escape 'BSe' where e is one of \'abfnrvt as per python escape sequences

hex-escape 'BSx[0-9a-fA-F][0-9a-fA-F]' - create an arbitrary character

2.4.4 tunestr literals

"char*" a sequence of character literals. Single quotes from the character literals are removed and the whole sequence is surrounded by double quotes (").

This literal is **ONLY** used for tune strings!

2.5 Variables

- On first assignment to a variable it's type is then set (int, list, tunestring) and it is an error to assign a different type to it. (this is different from Python but is needed for a compiled language)
- Integer variables are created when they are first assigned to:
  1. From another integer variable (eg. speed = velocity)
  2. From an integer literal (eg. speed = 34, speed = False)
  3. From an element of a list (eg. speed = code[x])
  4. From a function that returns integers (eg. speed = ord('a'))
- List variables are created when they are first assigned to:
  1. From the result of Ed.List()

  Note that the assignment List2 = List1 (where List1 is a list) just makes List2 an alias for List1
- TuneString variables are created when they are first assigned to:
  1. From the result of Ed.TuneString()

  Note that the assignment TS2 = TS1 (where TS1 is a tunestring) just makes TS2 an alias for TS1

2.6 Operators

-X, +X Negation and identity

X or Y, X and Y, not X logical or/and/not

X + Y, X - Y, X * Y Basic arithmetic

X / Y, X // Y, X % Y Divisions (both / and // return the nearest whole number less then the float result) Question - could Bill do this as the default?

X | Y, X & Y, X ^ Y, ~X bitwise or/and/xor/complement

X » Y, X « Y Shifts right and left

X = Y, X ! Y, X < Y, X <= Y, X > Y, X >= Y comparisons

2.7 Assignment

X op= expr (assignment to a simple variable)

X[index] op= expr (assignment to an item in a list or tune string). If the index is aconstant and beyond the end of the list, an error will be raised If index is a variable, no range checking is done

op is one of EMPTY|+|-|*|/|//|%|\||&|^|~| if op is not EMPTY, then same as X = X op Y

LIST2 = LIST1 or TUNESTRING1 = TUNESTRING1 Makes the LHS an alias for the RHS

LIST1 = list literal previous value of LIST1 is overwritten with list literal. If the list literal is larger then max size of LIST1, an OutOfRange error will be raised.

TUNESTRING1 = tune string literal previous value of TUNESTRING1 is overwritten with tune string literal. If the tune string literal is larger then max size of TUNESTRING1, an OutOfRange error will be raised.

2.8 Built in functions

abs(int) → int Returns the absolute value of the int

len(LIST) → int returns the MAX number of elements in the list

len(TUNESTR) → int returns the MAX number of characters in the tunestr (including final 'z')

ord(char) → int returns the ascii code for the char

chr(int) → char converts an ascii code into a char. NOTE - This will NOT be checked when the int is passed in a variable! The firmware though will stop playing the tune string if an invalid char is assigned to it.

2.9 Functions

- def NAME parameters : suite
- nested functions are NOT supported
- all parameters have to be used, there is no *args
- the compiler will check all calls to a function and make sure that all arguments are used consistently. All ints are passed by value, lists/tune strings are passed by reference
- global can be used to refer to global scope, all other variables are local ones
- calling functions – NAME([args])
- All 'return' statements must return the same type – nothing or int. Lists and tune strings don't have to be returned as they are passed by reference (so changes happen in the passed list/tunestring). It means that we can't have 'factory' functions creating lists or tunestrings. This is a GOOD thing with our restricted RAM!

2.10 Classes

- class NAME testlist : suite
- inheritance is NOT supported
- member functions are like functions listed above, but they have an extra argument, self, which refers to the object of this class

2.11 Control flow

- pass is supported to stand in for an empty suite

2.11.1 If statement

- if test: suite [elif test: suite]* [else: suite]

2.11.2 While statement

- while test: suite [else: suite]
- break and continue are supported

2.11.3 For statement

- for X in LIST|RANGE: suite
- for list this is functionally equivalent to: i = 0 while i < len(LIST): X = LIST[i]; i += 1; suite
- for range this is functionally equivalent to: range([START,] STOP [,STEP]) i = START (or 0 if not there) while i < STOP: X = i; i += STEP; suite
- break and continue are supported

2.12 Edison object

- import the 'Ed' module to access these functions
- The compiler uses the Ed module calls to generate code directly. A simulator could use the python in a real 'Ed' module to simulate running on an Edison
- When compiling Ed.Py, 'name' `= 'Ed.Py'. So can use code like if __name__ !` 'Ed.Py': – to execute non-Ed.Py code

1. Types / Variables
   - Ed.List(MaxElements [, INITIAL_LIST])
   - Ed.TuneString(MaxElements [, INITIAL_TUNE_STRING])
   - Ed.StrictTypes = True|False – if True then no conversion between types happens on function calls **Not sure this is still needed!**

   NOTE – historical notes on the design of the Edison object have been added to the end of the notes section of this document.

   Those notes have been superceded in a separate document created and maintained by Ben. It is on Redmine/TechnicalDocuments: "EdPy-Specification.docx". It contains the Edison object details and information about the text editor app.

2.13 Notes

2.13.1 Compatibility with python 2/3

It is important the Ed.Py doesn't teach bad habits, so Ed.Py is intentionally a subset of python. In fact all normal python keywords are planned on being recognised, even if not supported in Ed.Py. So it will be an error to use a python (not just Ed.Py) keyword as an identfier.

2.13.2 Import

I would have really like to import other files, as it could be helpful for sharing code in class. But, it makes the strategy of editing on a local machine, but compiling on the net difficult. So, a general import will not be supported (though 'import edison' will be used to signal that edison code is to be generated or simulated).

Life could be easier if support 'import edison as X', so something like 'import edison as e' could be done. But this would make the webapp harder to provide 'intellisense'

2.13.3 Concurrency

Could improve the stack to help with local variables, or could use the compiler to provide two sets of local variables, if it's called from both an event handler and from normal code

Question – verify with Bill that only 1 event handler can run at a time!

2.13.4 OLD historical edison object notes

Note – the document EdPy-Specification.docx supersedes this! This is here for historical purposes only.

1. Simple control
   - edison.control_led(WHICH, STATE)
     - constants: edison.LED_LEFT, edison.LED_RIGHT, edison.LED_ON, edison.LED_OFF
   - edison.play_beep()
   - edison.play_my_beep(FREQ_CODE, DUR_CODE) - with FREQ_CODE = 5529600 / desired freq in Hz. DUR_CODE is time in ms / 10.
   - edison.play_note(NOTE, DURATION)
     - constants: edison. - NOTE_A_6, NOTE_B_SHARP_6, NOTE_B_6, NOTE_C_7, NOTE_D_SHARP_7, NOTE_D_7, NOTE_E_SHARP_7, NOTE_E_7, NOTE_F_7, NOTE_G_SHARP_7, NOTE_G_7, NOTE_A_SHARP_7, NOTE_A_7 NOTE_B_SHARP_7, NOTE_B_7, NOTE_C_8, NOTE_REST NOTE_SIXTEENTH, NOTE_QUARTER, NOTE_HALF, NOTE_WHOLE
   - edison.play_tune(BYTE_TUNE_LIST)
     - constants: edison. - TUNE_A_6, TUNE_A_SHARP_6, TUNE_B_6, TUNE_C_7, TUNE_C_SHARP_7, TUNE_D_SHARP_7, TUNE_D_7, TUNE_E_7, TUNE_F_7, TUNE_F_SHARP_7, TUNE_G_SHARP_ TUNE_G_7, TUNE_A_SHARP_7, TUNE_A_7 TUNE_B_7, TUNE_C_8, TUNE_REST, TUNE_0_20TH TUNE_1_20TH, TUNE_2_20TH, TUNE_3_20TH, TUNE_4_20TH, TUNE_5_20TH TUNE_6_20TH, TUNE_7_20TH
   - NEW edison.play_music(INT_MUSIC_LIST) -constants: TBD
     - Question do we play music
   - edison.control_detection(STATE)
     - constants: edison.DETECTION_OFF, edison.DETECTION_ON
   - edison.drive_motor(WHICH, DIR, SPEED, DIST)

  - constants: edison.LEFT_MOTOR, edison.RIGHT_MOTOR, edison.DRIVE_FORWARD, edison.DRIVE_BAC
    edison.DRIVE_STOP, edison.DRIVE_SPEED_0, . . . , edison.DRIVE_SPEED_10, edison.DIST_UNLIMITED
    edison.DIST_7_5_DEGREES_MULT, edison.DIST_2_5_MM_MULT, edison.DIST_INCH_MULT
  - question: distance is now included?
- edison.drive_pair(DIR, SPEED, DISTANCE)
  - constants from edison.drive_motor(), but without edison.XXXX_MOTOR
  - question: distance is now included?
- edison.drive_pair_trick(TRICK, SPEED, DISTANCE)
  - constants from edison.drive_pair(), but without edison.DRIVE_XXXX
  - constants: edison.TRICK_FORWARD_RIGHT, edison.TRICK_FORWARD_LEFT, edison.TRICK_BACKW
    edison.TRICK_BACKWARD_LEFT, edison.TRICK_FORWARD_SPIN_RIGHT, edison.TRICK_FORWARI
    edison.TRICK_BACKWARD_SPIN_RIGHT, edison.TRICK_BACKWARD_SPIN_LEFT
  - question: distance is now included?
- edison.control_tracker(STATE)
  - constants: edison.LINE_TRACKER_ON, edison.LINE_TRACKER_OFF
- edison.send_byte(BYTE_DATA)
  - constants: none
- edison.start_count_down(INT_START_VALUE)
  - constants: none

2. Read
   - BYTE = edison.read_detection()
     - constants: edison.OBSTACLE_NONE, edison.OBSTACLE_DETECTED, edison.OBSTACLE_RIGHT, edison.OBSTACLE_LEFT, edison.OBSTACLE_AHEAD
   - BYTE = edison.read_keypad()
     - constants: edison.KEYPAD_NONE, edison.KEYPAD_TRIANGE, edison.KEYPAD_ROUND
   - BYTE = edison.read_clap()
     - constants: edison.CLAP_NOT_DETECTED, edison.CLAP_DETECTED
   - BYTE = edison.read_line()
     - constants: edison.LINE_ON_BLACK, edison.LINE_ON_WHITE
   - BYTE = edison.read_remote() - returns last received remote control command
     - constants: none
   - BYTE = edison.read_data() - returns last received infrared data
     - constants: none
   - INT = edison.read_light_level(WHICH)
     - constants: edison.LIGHT_LEVEL_LEFT, edison.LIGHT_LEVEL_RIGHT, edison.LIGHT_LEVEL_LINE_T
   - INT = edison.read_count_down()
     - constants: none
3. Data
   - These are all taken care of with normal python expressions and assignments
   - +1, -1, set memory, copy data, calculate all handled with assignments and expressions
4. Flow
   - loop and if/else are handled with python while/for/if/elif/else
   - edison.time_wait(TICKS)
     - constants: edison.TIME_10_MS_MULT
   - edison.event_wait(SOURCE, STATE) -constants: see the constants in the next section
5. Events
   - each event function is named like this: edison.register_event_handler(SOURCE, STATE, functionName)

- constants: edison.EVENT_SOURCE_COUNTDOWN_TIMER, edison.EVENT_SOURCE_REMOTE, edison.EVENT_SOURCE_DATA, edison.EVENT_SOURCE_DETECT_CLAP, edison.EVENT_SOURCE_D edison.EVENT_SOURCE_DRIVE, edison.EVENT_SOURCE_KEYPAD, edison.EVENT_SOURCE_LINE_ edison.EVENT_SOURCE_MUSIC, edison.EVENT_TIMER_FINISHED, edison.EVENT_REMOTE_CODE_ edison.EVENT_IR_DATA, edison.EVENT_CLAP_DETECTED, edison.EVENT_OBSTACLE_ANY, edison.EVENT_OBSTACLE_LEFT, edison.EVENT_OBSTACLE_RIGHT, edison.EVENT_OBSTACLE_AH edison.EVENT_DRIVE_STRAIN, edison.EVENT_KEYPAD_TRIANGLE, edison.EVENT_KEYPAD_ROU edison.EVENT_LINE_TRACKER_ON_WHITE, edison.EVENT_LINE_TRACKER_ON_BLACK, edison.EVENT_LINE_TRACKER_SURFACE_CHANGE, edison.EVENT_TUNE_FINISHED, edison.EVENT_MUSIC_FINISHED

6. Low Level

- common constants: edison.MODULE_LINE_TRACKER, edison.MODULE_LEFT_LED, edison.MODULE_RIGH edison.MODULE_LEFT_DRIVE, edison.MODULE_RIGTH_DRIVE, edison.MODULE_IR_RX, edison.MODULE_IR_TX, edison.MODULE_BEEPER
- edison.read_byte_module_register(MOD, REG)
- edison.write_byte_module_register(MOD, REG, NEW_VALUE)
- edison.read_int_module_register(MOD, REG)
- edison.write_int_module_register(MOD, REG, NEW_VALUE)