Edison

EdPy app documentation

This document contains a full copy of the help text content available in the ‘Documentation’ section of the EdPy app.

microbric
motivate • create • educate

Document number: 3.2.2.18.3 Rev3

Contents

Ed.List() 4
Ed.LeftLed() 5
Ed.RightLed() 6
Ed.ObstacleDetectionBeam() 7
Ed.LineTrackerLed() 8
Ed.SendIRData() 9
Ed.StartCountDown() 10
Ed.TimeWait() 11
Ed.RegisterEventHandler() 12
Ed.PlayBeep() 14
Ed.PlayMyBeep() 15
Ed.PlayTone() 16
Ed.PlayTune() 18
Ed.TuneString() 20
Ed.Drive() 23
Ed.DriveLeftMotor() 26
Ed.DriveRightMotor() 29
Ed.ReadObstacleDetection() 32
Ed.ReadKeypad() 34
Ed.ReadClapSensor() 35
Ed.ReadLineState() 37
Ed.ReadRemote() 38
Ed.ReadIRData() 40
Ed.ReadLeftLightLevel() 41
Ed.ReadRightLightLevel() 42
Ed.ReadLineTracker() 43
Ed.ReadCountDown() 44

Ed.ReadMusicEnd() 45
Ed.ReadDriveLoad() 46
Ed.ReadDistance() 47
Ed.SetDistance() 48
Ed.ResetDistance() 49
Ed.ON 50
Ed.OFF 50
Ed.NOTE_# 50
Ed.TEMPO_# 52
Ed.FORWARD 53
Ed.BACKWARD 53
Ed.FORWARD_RIGHT 54
Ed.BACKWARD_RIGHT 54
Ed.FORWARD_LEFT 54
Ed.BACKWARD_LEFT 55
Ed.SPIN_# 55
Ed.STOP 55
Ed.SPEED_# 56
Ed.DISTANCE_UNLIMITED 57
Ed.MOTOR_# 57
Ed.TIME_# 58
Ed.OBSTACLE_# 58
Ed.LINE_ON_# 59
Ed.KEYPAD_# 59
Ed.CLAP_# 60
Ed.DRIVE_# 60
Ed.MUSIC_# 61
Ed.REMOTE_CODE_# 61
Ed.EVENT_# 62
Ed.CM 64
Ed.INCH 64
Ed.TIME 64
Ed.V1 65

Ed.V2 65
Ed.Tempo 65
Ed.DistanceUnits 66
Ed.EdisonVersion 68
abs() 69
len() 70
range() 71
True 72
False 72
import 73
def 73
pass 75
for 76
while 77
if/elif/else 78
class 79
return 82

Ed.List()

Ed.List(size)

Parameters:

Size:

A positive integer - sets the number of integers in the new list.

The maximum size is 250 integers.

Returns:

A list object.

Ed.List(size,initialList)

Parameters:

Size:

A positive integer - sets the number of integers in the new list.

The maximum size is 250 integers.

Initial List:

A python style list e.g. [1,2,3] - sets the initial value of the integers in the new Ed.List.

Returns:

A list object.

Explanation:

Creates a list of Edison variables.

Examples:

Create an empty list and fill it with zeros.

```
#--------Your code below-----------

zeros=Ed.List(5)

for x in range(5):

    zeros[x]=0
```

Create a new list with pre-filled values.

```
#--------Your code below-----------

example=Ed.List(5,[1,2,3,4,5])
```

Watch out for:

The maximum list size is 250.

Additional new elements cannot be added to the end of the list. The list is a fixed size. Python lists are "0 index" lists, meaning the first element in the list is at index 0. For example, using the pre-filled list from the above example, the following code would flash Edison's LED once.

```
while example[0]!=0:

    Ed.LeftLed(Ed.ON)

    Ed.TimeWait(500, Ed.TIME_MILLISECONDS)

    Ed.LeftLed(Ed.OFF)

    Ed.TimeWait(500, Ed.TIME_MILLISECONDS)

    example[0]=example[0]-1
```

Ed.LeftLed()

Ed.LeftLed(state)

Parameters:

State:

- Ed.ON – LED turns on
- Ed.OFF – LED turns off

Returns:

N/A

Explanation:

Turns Edison's left LED on or off.

Examples:

Quick flash of left LED.

```
#--------Your code below-----------

Ed.LeftLed(Ed.ON)

Ed.TimeWait(500, Ed.TIME_MILLISECONDS)

Ed.LeftLed(Ed.OFF)

Ed.TimeWait(500, Ed.TIME_MILLISECONDS)
```

Left LED on while driving.

```
#--------Your code below-----------

Ed.LeftLed(Ed.ON)

Ed.Drive(Ed.FORWARD, Ed.SPEED_5, 10)

Ed.LeftLed(Ed.OFF)
```

Watch out for:

If used to turn Edison's LED on, another function call is needed later in the code to turn Edison's LED off.

Ed.RightLed()

Ed.RightLed(state)

Parameters:

State:

- Ed.ON – LED turns on
- Ed.OFF – LED turns off

Returns:

N/A

Explanation:

Turns Edison's right LED on or off.

Examples:

Quick flash of right LED.

```
#--------Your code below-----------

Ed.RightLed(Ed.ON)

Ed.TimeWait(500, Ed.TIME_MILLISECONDS)

Ed.RightLed(Ed.OFF)

Ed.TimeWait(500, Ed.TIME_MILLISECONDS)
```

Right LED on while driving.

```
#--------Your code below-----------

Ed.RightLed(Ed.ON)

Ed.Drive(Ed.FORWARD, Ed.SPEED_5, 10)

Ed.RightLed(Ed.OFF)
```

Watch out for:

If used to turn Edison's LED on, another function call is needed later in the code to turn Edison's LED off.

Ed.ObstacleDetectionBeam()

Ed.ObstacleDetectionBeam(state)

Parameters:

State:

- Ed.ON – Obstacle detection functions are enabled.

- Ed.OFF – Obstacle detection functions are disabled.

Returns:

N/A

Explanation:

Turns Edison's obstacle detection IR system on or off. This is required to use other obstacle detection functions.

Examples:

Turn on the obstacle detection beam and beep at obstacles.

```
#--------Your code below-----------

Ed.ObstacleDetectionBeam(Ed.ON)

while True:

    if Ed.ReadObstacleDetection()>Ed.OBSTACLE_NONE:

        Ed.PlayBeep()
```

Watch out for:

While the obstacle detection beam needs to be turned on to enable Edison to detect an obstacle, this function is not used to detect obstacles. Use Ed.ReadObstacleDetection() to have Edison react to obstacles.

Edison's obstacle detection and IR messaging functions both use the same IR LEDs and IR receiver. Therefore, Edison cannot send or receive messages from other Edison robots if the obstacle detection beam is turned on.

Ed.LineTrackerLed()

Ed.LineTrackerLed(state)

Parameters:

State:

- Ed.ON – LED turns on
- Ed.OFF – LED turns off

Returns:

N/A

Explanation:

Turns Edison's line tracker LED on or off. This is required to use other line tracking functions.

Examples:

Turn on the line tracking LED and beep when a black surface is detected.

```
#--------Your code below-----------

Ed.LineTrackerLed(Ed.ON)

while True:

    if Ed.ReadLineState()==Ed.LINE_ON_BLACK:

        Ed.PlayBeep()
```

Watch out for:

Always start the Edison robot on a white surface when running a program with this function. When Edison turns on the line tracking LED, a reading from the line tracking sensor is taken. This first reading is set to be a white surface and the Ed.ReadLineState() function uses this as a baseline. If the line tracking LED is turned on while the robot is over a black line, this will cause an error where Edison cannot find something which reflects less light, and will therefore never return the LINE_ON_BLACK condition.

While the line tracking LED needs to be turned on to enable Edison to detect a line, this function is not used to react to lines. Use Ed.ReadLineState() to have Edison react to lines.

Ed.SendIRData()

Ed.SendIRData(byte)

Parameters:

Byte:

A positive integer between 0-255 to send to all nearby Edison robots.

Returns:

N/A

Explanation:

Sends one byte of data out to be received by other Edison robots via infrared.

Examples:

Send a simple value.

```
#--------Your code below-----------

Ed.SendIRData(10)
```

Send line tracking data.

```
#--------Your code below-----------

Ed.LineTrackerLed(Ed.ON)

lineState = Ed.ReadLineState()

Ed.SendIRData(lineState)
```

Watch out for:

Only 8-bit variables (range of 0-255) can be sent. Since EdPy uses 16-bit variables, this function ignores the top 8 bits of any input.

Ed.StartCountDown()

Ed.StartCountDown(time, units)

Parameters:

Time:

The number of seconds or milliseconds to count down from.

The maximum value is 32767.

Units:

- Ed.TIME_MILLISECONDS – Counts down in milliseconds.
- Ed.TIME_SECONDS – Counts down in seconds.

Returns:

N/A

Explanation:

Set the countdown timer to a set number of seconds or milliseconds. The counter then starts to count down in the background.

Examples:

Flash the left LED for 3000 milliseconds.

```
#--------Your code below-----------

Ed.StartCountDown(3100, Ed.TIME_MILLISECONDS);

while Ed.ReadCountDown(Ed.TIME_MILLISECONDS) > 100:

    Ed.LeftLed(Ed.ON)

    Ed.TimeWait(50, Ed.TIME_MILLISECONDS)

    Ed.LeftLed(Ed.OFF)

    Ed.TimeWait(50, Ed.TIME_MILLISECONDS)
```

Watch out for:

The timer can ONLY count down, not up. It cannot be used as a stopwatch counting up, only as a timer counting down.

The timer is not a clock and cannot tell time.

Ed.TimeWait()

Ed.TimeWait(time, units)

Parameters:

Time:

A positive integer number of seconds or milliseconds to wait for.

The maximum value is 32767.

Units:

- Ed.TIME_MILLISECONDS – Counts down in milliseconds.
- Ed.TIME_SECONDS – Counts down in seconds.

Returns:

N/A

Explanation:

Stops the program from continuing until it waits for the specified amount of time to pass.

Examples:

Turn on the left LED for three seconds, off for half a second and then back on.

```
#--------Your code below-----------

Ed.LeftLed(Ed.ON)

Ed.TimeWait(3, Ed.TIME_SECONDS)

Ed.LeftLed(Ed.OFF)

Ed.TimeWait(500, Ed.TIME_MILLISECONDS)

Ed.LeftLed(Ed.ON)
```

Watch out for:

Time can only be set as an integer value. To wait for a period less than one second, use Ed.TIME_MILLISECONDS as the units. As an example, 3500 milliseconds are equal to 3.5 seconds.

Ed.RegisterEventHandler()

Ed.RegisterEventHandler(event, function)

Parameters:

State:

- Ed.EVENT_TIMER_FINISHED - Calls the function when the countdown timer finishes.

- Ed.EVENT_REMOTE_CODE - Calls the function when Edison receives a remote code.
- Ed.EVENT_IR_DATA - Calls the function when Edison receives code from another Edison.
- Ed.EVENT_CLAP_DETECTED - Calls the function when Edison detects a clap.
- Ed.EVENT_OBSTACLE_ANY - Calls the function when Edison detects any obstacle.
- Ed.EVENT_OBSTACLE_LEFT - Calls the function when Edison detects an obstacle to the left.
- Ed.EVENT_OBSTACLE_RIGHT - Calls the function when Edison detects an obstacle to the right.
- Ed.EVENT_OBSTACLE_AHEAD - Calls the function when Edison detects an obstacle straight ahead.
- Ed.EVENT_DRIVE_STRAIN - Calls the function when Edison detects strain on the drive.
- Ed.EVENT_KEYPAD_TRIANGLE - Calls the function when Edison detects a triangle button press.
- Ed.EVENT_KEYPAD_ROUND - Calls the function when Edison detects a round button press.
- Ed.EVENT_LINE_TRACKER_ON_WHITE - Calls the function when Edison detects a white surface under the line tracker.
- Ed.EVENT_LINE_TRACKER_ON_BLACK - Calls the function when Edison detects a black surface under the line tracker.
- Ed.EVENT_LINE_TRACKER_SURFACE_CHANGE - Calls the function when Edison detects a surface change under the line tracker.
- Ed.EVENT_TUNE_FINISHED - Calls the function when Edison finishes playing a tune.

Function:

The string name of a user-created function to be called when an event occurs.

Returns:

N/A

Explanation:

Sets up an ‘event handler’, causing Edison to call a function when a given event occurs.

Examples:

Flash the left LED forever and beep whenever an obstacle is detected.

```
#--------Your code below-----------
```

```
Ed.ObstacleDetectionBeam(Ed.ON)

Ed.RegisterEventHandler(Ed.EVENT_OBSTACLE_ANY, "whenObsBeep")

while True:

    Ed.LeftLed(Ed.ON)

    Ed.TimeWait(500, Ed.TIME_MILLISECONDS)

    Ed.LeftLed(Ed.OFF)

    Ed.TimeWait(500, Ed.TIME_MILLISECONDS)

def whenObsBeep():

    Ed.PlayBeep()
```

Watch out for:

Event handlers act as an interrupt, which means that when the event occurs, the main program is paused while the given function is run. When the function completes, the main program continues where it left off.

Ed.PlayBeep()

Ed.PlayBeep()

Parameters:

N/A

Returns:

N/A

Explanation:

Sounds a single beep with frequency: 3.5KHz, duration: 50ms (0.05 seconds).

Examples:

Play a beep.

```
#--------Your code below-----------
```

```
Ed.PlayBeep()
```

Watch out for:

All of Edison's sounds occur in the background. As such, Edison moves onto the next line of code as soon as the sound starts without waiting for the sound to finish. To make Edison wait for the sound to finish, use the Ed.ReadMusicEnd() function in a loop.

Ed.PlayMyBeep()

Ed.PlayMyBeep(period)

Parameters:

Period:

The period is how long it takes an acoustic wave to complete a full cycle. Changing this number causes a change in the frequency of the sound played because when period increases, frequency decreases. To convert a frequency into a period, divide the number 8,000,000 by the desired frequency.

For example, to play a 1kHz (1000 cycles per second) sound: 8,000,000/1,000 = 8,000

Returns:

N/A

Explanation:

Sounds a single beep with a given period for a duration of 50ms (0.05 seconds).

Examples:

Play a beep at 1Khz (8000 period).

```
#----------Your code below--------------

Ed.PlayMyBeep(8000)
```

Watch out for:

All of Edison's sounds occur in the background. As such, Edison moves onto the next line of code as soon as the sound starts without waiting for the sound to finish. To make Edison wait for the sound to finish, use the Ed.ReadMusicEnd() function in a loop.

The largest number Edison can use is 32767, which means Edison cannot handle the number 8000000 which is required to convert frequency to period. Therefore, you will need to calculate the period you want before programming Edison and use this as the number in your program.

Ed.PlayTone()

Ed.PlayTone(note, duration)

Parameters:

Note:

- Ed.NOTE_A_6 - Play a low A.
- Ed.NOTE_A_SHARP_6 - Play a low A sharp.
- Ed.NOTE_B_6 - Play a low B.
- Ed.NOTE_C_7 - Play a C.
- Ed.NOTE_C_SHARP_7 - Play a C sharp.
- Ed.NOTE_D_7 - Play a D.
- Ed.NOTE_D_SHARP_7 - Play a D sharp.
- Ed.NOTE_E_7 - Play an E.
- Ed.NOTE_F_7 - Play an F.
- Ed.NOTE_F_SHARP_7 - Play an F sharp.
- Ed.NOTE_G_7 - Play a G.
- Ed.NOTE_G_SHARP_7 - Play a G sharp.
- Ed.NOTE_A_7 - Play an A.
- Ed.NOTE_A_SHARP_7 - Play an A sharp.
- Ed.NOTE_B_7 - Play a B.
- Ed.NOTE_C_8 - Play a high C.
- Ed.NOTE_REST - Play a rest.
- A positive integer representing period. The period is how long it takes an acoustic wave to complete a full cycle. Changing this number causes a change in the frequency of the sound played because when period increases, frequency decreases. To convert a frequency into a period, divide the number

8,000,000 by the desired frequency. For example, to play a 1kHz (1000 cycles per second) sound: 8,000,000/1,000 = 8,000

***Duration*:**

- Ed.NOTE_SIXTEENTH - Play note for 125 milliseconds.
- Ed.NOTE_EIGHTH - Play note for 250 milliseconds.
- Ed.NOTE_QUARTER - Play note for 500 milliseconds.
- Ed.NOTE_HALF - Play note for 1000 milliseconds.
- Ed.NOTE_WHOLE - Play note for 2000 milliseconds.
- A positive integer between 0 -32767 representing duration in milliseconds.

Returns:

N/A

Explanation:

Sounds a single note with a given period, for a given length of time.

Examples:

Play a 1Khz note (8000 period) for two seconds (2000 milliseconds).

```
#-----------Your code below---------------

Ed.PlayTone(8000, 2000)

while Ed.ReadMusicEnd()==Ed.MUSIC_NOT_FINISHED:

    pass
```

Play an A sharp for half a second.

```
#-----------Your code below---------------

Ed.PlayTone(Ed.NOTE_A_SHARP_6, Ed.NOTE_QUARTER)

while Ed.ReadMusicEnd()==Ed.MUSIC_NOT_FINISHED:

    pass
```

Watch out for:

All of Edison's sounds occur in the background. As such, Edison moves onto the next line of code as soon as the sound starts without waiting for the sound to finish.

To make Edison wait for the sound to finish, use the Ed.ReadMusicEnd() function in a loop.

The largest number Edison can use is 32767, which means Edison cannot handle the number 8000000 which is required to convert frequency to period. Therefore, you will need to calculate the period you want before programming Edison and use this as the number in your program.

To determine the period to use to exactly match the Ed.NOTE constants, divide the number 32,000,000 by the desired frequency of that musical note and use the result as the note input parameter.

Ed.PlayTune()

Ed.PlayTune(Tune)

Parameters:

Tune:

Takes an Edison tune, which is a Python-style string which needs to be created using the Ed.TuneString() function.

A tune string looks like this: "ndndndndndnd...ndz" where n is a note from the notes table, and d is duration from the duration table – see below.

Notes:

m - low A

M - low sharp A

n - low B

c - C

C - C sharp

d - D

D - D sharp

e - E

f - F

F - F sharp

g - G

G - G sharp

a - A

A - A sharp

b - B

o - high C

Duration:

1 - whole note

2 - half note

4 - quarter note

8 - eighth note

6 - sixteenth note

Other:

R - rest

z - end of tune

Returns:

N/A

Explanation:

Plays a string of musical notes through the speaker. This is done by passing the function a 'tune string' made up of a string of notes using the tables above as a reference using the Ed.TuneString() function. You can change the speed you tune plays by changing what Ed.Tempo is set to in the setup code.

Examples:

Play a simple tune.

```
#--------Your code below-----------

simple = Ed.TuneString(25, "d4e4f4e4d4c4n2d4e4f4e4d1z")

Ed.PlayTune(simple)
```

```
while Ed.ReadMusicEnd()==Ed.MUSIC_NOT_FINISHED:

    pass
```

Watch out for:

The function must use a Python-style string which needs to be created using the Ed.TuneString() function.

All tune strings need to end with a "z" character to end correctly.

All of Edison's sounds occur in the background. As such, Edison moves onto the next line of code as soon as the sound starts without waiting for the sound to finish. To make Edison wait for the sound to finish, use the Ed.ReadMusicEnd() function in a loop.

You can change the speed you tune plays by changing what Ed.Tempo is set to in the setup code.

Ed.TuneString()

Ed.TuneString(size)

Parameters:

Size:

A positive integer - sets the number of characters in the new string.

The maximum size is 250 integers.

Returns:

A python-style string of notes.

Ed.TuneString(size,initialTune)

Parameters:

Size:

A positive integer - sets the number of characters in the new string.

The maximum size is 250 integers.

InitialTune:

A Python-style string, for example "a4g2z", which sets the notes to be played in the tune.

A tune string looks like this: "ndndndndndnd...ndz" where n is a note from the notes table, and d is duration from the duration table – see below.

Notes:

m - low A

M - low sharp A

n - low B

c - C

C - C sharp

d - D

D - D sharp

e - E

f - F

F - F sharp

g - G

G - G sharp

a - A

A - A sharp

b - B

o - high C

Duration:

1 - whole note

2 - half note

4 - quarter note

8 - eighth note

6 - sixteenth note

Other:

R - rest

z - end of tune

Returns:

A python-style string of notes.

Explanation:

Creates a new tune string which can be used with the Ed.PlayTune() function.

Examples:

Play a simple tune.

```
#--------Your code below-----------

simple = Ed.TuneString(25, "d4e4f4e4d4c4n2d4e4f4e4d1z")

Ed.PlayTune(simple)

while Ed.ReadMusicEnd()==Ed.MUSIC_NOT_FINISHED:

    pass
```

Watch out for:

The function will not play a tune, but only creates a Python-style string. To play the tune, use the Ed.PlayTune() function.

The maximum tune size is 250 characters.

All tune strings need to end with a "z" character to end correctly.

All of Edison's sounds occur in the background. As such, Edison moves onto the next line of code as soon as the sound starts without waiting for the sound to finish. To make Edison wait for the sound to finish, use the Ed.ReadMusicEnd() function in a loop.

You can change the speed you tune plays by changing what Ed.Tempo is set to in the setup code.

Ed.Drive()

Ed.Drive(direction,speed,distance)

Parameters:

Direction:

- Ed.FORWARD - Edison drives forwards.
- Ed.BACKWARD - Edison drives backwards.
- Ed.FORWARD_RIGHT - Edison uses one wheel to turn forwards right (clockwise).
- Ed.BACKWARD_RIGHT - Edison uses one wheel to turn backwards right (counter-clockwise).
- Ed.FORWARD_LEFT - Edison uses one wheel to turn forwards left (counter-clockwise).
- Ed.BACKWARD_LEFT - Edison uses one wheel to turn backwards left (clockwise).
- Ed.SPIN_RIGHT - Edison spins on the spot to the right (clockwise).
- Ed.SPIN_LEFT - Edison spins on the spot to the left (counter-clockwise).
- Ed.STOP - Stops Edison immediately.

Speed:

- A positive integer number between 1-10.
- Ed.SPEED_1 - PWM controlled speed setting 1.
- Ed.SPEED_2 - PWM controlled speed setting 2.
- Ed.SPEED_3 - PWM controlled speed setting 3.
- Ed.SPEED_4 - PWM controlled speed setting 4.
- Ed.SPEED_5 - PWM controlled speed setting 5.
- Ed.SPEED_6 - PWM controlled speed setting 6.
- Ed.SPEED_7 - PWM controlled speed setting 7.
- Ed.SPEED_8 - PWM controlled speed setting 8.
- Ed.SPEED_9 - PWM controlled speed setting 9.
- Ed.SPEED_10 - PWM controlled speed setting 10.
- Ed.SPEED_FULL - Full power to the motors. (Please note- Edison may not drive perfectly straight with this setting.)

Distance:

- An integer number for distance. The maximum value is 32767.

OR

- Ed.DISTANCE_UNLIMITED - turns on Edison's motors and moves on with the code. (Note: a stop will be needed later in the code.)

If using an integer number, note that the unit value of this number is set by Ed.DistanceUnits in the setup code. You can change the unit value by changing what Ed.DistanceUnits is set to in the setup code.

- Ed.DistanceUnits = Ed.CM - distance in centimetres (default for V2.0).
- Ed.DistanceUnits = Ed.INCH - distance in inches.
- Ed.DistanceUnits = Ed.TIME - distance in milliseconds (default for V1).

Note: When Edison is turning and Ed.DistanceUnits is set to CM or INCH, distance becomes the number of degrees to turn with a maximum value of 360.

Returns:

N/A

Explanation:

Controls both of Edison's motors to create movement. This can be set to move for a set distance (CM or INCH) or time period (TIME) and will drive the full distance before moving onto the next line of code.

Examples:

Drive Edison forward for 3 cm at speed 5. (Edison V2.0 only)

```
Ed.DistanceUnits = Ed.CM

Ed.Tempo = Ed.TEMPO_MEDIUM

#--------Your code below-----------

Ed.Drive(Ed.FORWARD, Ed.SPEED_5, 3)
```

Drive Edison forward for 5 inches at speed 5. (Edison V2.0 only)

```
Ed.DistanceUnits = Ed.INCH

Ed.Tempo = Ed.TEMPO_MEDIUM

#--------Your code below-----------

Ed.Drive(Ed.FORWARD, Ed.SPEED_5, 5)
```

Drive Edison forward for 2000 milliseconds at speed 7.

```
Ed.DistanceUnits = Ed.TIME

Ed.Tempo = Ed.TEMPO_MEDIUM

#--------Your code below-----------

Ed.Drive(Ed.FORWARD, Ed.SPEED_7, 2000)
```

Spin Edison left 90 degrees at speed 10.

```
Ed.DistanceUnits = Ed.CM

Ed.Tempo = Ed.TEMPO_MEDIUM

#--------Your code below-----------

Ed.Drive(Ed.SPIN_LEFT, 10, 90)
```

Set speed and distance with variables.

```
Ed.DistanceUnits = Ed.CM

Ed.Tempo = Ed.TEMPO_MEDIUM

#--------Your code below-----------

driveSpeed=5

driveDistance=10

Ed.Drive(Ed.FORWARD, driveSpeed, driveDistance)
```

Watch out for:

Distance units of CM and INCH are only available for Edison V2.0. If you are using a V1 Edison, please make sure that Ed.DistanceUnits = Ed.TIME.

The Ed.TIME constant is in milliseconds. When Ed.DistanceUnits = Ed.TIME, remember that the distance input integer is milliseconds. Example: to drive for 2 seconds, a distance of 2000 should be input.

When distance is set to Ed.FORWARD_RIGHT, Ed.FORWARD_LEFT or Ed.SPIN_# and Ed.DistanceUnits is set to CM or INCH, any distance input will instead be considered the number of degrees that Edison will turn up to a maximum value of 360. If you put in a value above 360, the code will wrap the value around and Edison will perform a shorter turn. Example: an input of 380 will cause Edison to turn 20 degrees.

Ed.SPEED_FULL turns Edison's motors up to the maximum value. As such Edison has no control over the speed and Edison V2.0s will not be able to use their encoders to correct for driving drift.

Setting distance to 0 or Ed.DISTANCE_UNLIMITED makes Edison turn on the motors and move on with the code. An additional Ed.Drive() will be required later in the code to stop or change the direction of movement.

When the distance input is set to anything other than 0 or Ed.DISTANCE_UNLIMITED, Edison will drive for the full distance supplied before moving on to the next line of code.

Ed.DriveLeftMotor()

Ed.DriveLeftMotor(direction,speed,distance)

Parameters:

Direction:

- Ed.FORWARD - Edison's left motor drives forwards.
- Ed.BACKWARD - Edison's left motor drives backwards.
- Ed.STOP - Stops Edison's left motor immediately.

Speed:

- A positive integer number between 1-10.
- Ed.SPEED_1 - PWM controlled speed setting 1.
- Ed.SPEED_2 - PWM controlled speed setting 2.
- Ed.SPEED_3 - PWM controlled speed setting 3.
- Ed.SPEED_4 - PWM controlled speed setting 4.
- Ed.SPEED_5 - PWM controlled speed setting 5.
- Ed.SPEED_6 - PWM controlled speed setting 6.
- Ed.SPEED_7 - PWM controlled speed setting 7.
- Ed.SPEED_8 - PWM controlled speed setting 8.
- Ed.SPEED_9 - PWM controlled speed setting 9.

- Ed.SPEED_10 - PWM controlled speed setting 10.
- Ed.SPEED_FULL - Full power to the motors. (Please note- Edison may not drive perfectly straight with this setting.)

Distance:

- An integer number for distance. The maximum value is 32767.

OR

- Ed.DISTANCE_UNLIMITED - turns on Edison's motors and moves on with the code. (Note: a stop will be needed later in the code.)

If using an integer number, note that the unit value of this number is set by Ed.DistanceUnits in the setup code. You can change the unit value by changing what Ed.DistanceUnits is set to in the setup code.

- Ed.DistanceUnits = Ed.CM - distance in centimetres (default for V2.0).
- Ed.DistanceUnits = Ed.INCH - distance in inches.
- Ed.DistanceUnits = Ed.TIME - distance in milliseconds (default for V1).

Returns:

N/A

Explanation:

Controls Edison's left motor to create movement. This can be set to move for a set distance (CM or INCH) or time period (TIME) and will drive the full distance before moving onto the next line of code.

Examples:

Drive Edison's left motor forward for 3 cm at speed 5. (Edison V2.0 only)

```
Ed.DistanceUnits = Ed.CM

Ed.Tempo = Ed.TEMPO_MEDIUM

#--------Your code below-----------

Ed.DriveLeftMotor(Ed.FORWARD, Ed.SPEED_5, 3)
```

Drive Edison's left motor forward for 5 inches at speed 5. (Edison V2.0 only)

```
Ed.DistanceUnits = Ed.INCH

Ed.Tempo = Ed.TEMPO_MEDIUM
```

```
#--------Your code below-----------

Ed.DriveLeftMotor(Ed.FORWARD, Ed.SPEED_5, 5)
```

Drive Edison's left motor forward for 2000 milliseconds at speed 7.

```
Ed.DistanceUnits = Ed.TIME

Ed.Tempo = Ed.TEMPO_MEDIUM

#--------Your code below-----------

Ed.DriveLeftMotor(Ed.FORWARD, Ed.SPEED_7, 2000)
```

Set speed and distance with variables.

```
Ed.DistanceUnits = Ed.CM

Ed.Tempo = Ed.TEMPO_MEDIUM

#--------Your code below-----------

driveSpeed=5

driveDistance=10

Ed.DriveLeftMotor(Ed.FORWARD, driveSpeed, driveDistance)
```

Watch out for:

Distance units of CM and INCH are only available for Edison V2.0. If you are using a V1 Edison, please make sure that Ed.DistanceUnits = Ed.TIME.

The Ed.TIME constant is in milliseconds. When Ed.DistanceUnits = Ed.TIME, remember that the distance input integer is milliseconds. Example: to drive for 2 seconds, a distance of 2000 should be input.

When distance is set to Ed.FORWARD_RIGHT, Ed.FORWARD_LEFT or Ed.SPIN_# and Ed.DistanceUnits is set to CM or INCH, any distance input will instead be considered the number of degrees that Edison will turn up to a maximum value of 360. If you put in a value above 360, the code will wrap the value around and Edison will perform a shorter turn. Example: an input of 380 will cause Edison to turn 20 degrees.

Ed.SPEED_FULL turns Edison's motors up to the maximum value. As such Edison has no control over the speed and Edison V2.0s will not be able to use their encoders to correct for driving drift.

Setting distance to 0 or Ed.DISTANCE_UNLIMITED makes Edison turn on the motors and move on with the code. An additional Ed.Drive() will be required later in the code to stop or change the direction of movement.

When the distance input is set to anything other than 0 or Ed.DISTANCE_UNLIMITED, Edison will drive for the full distance supplied before moving on to the next line of code.

Ed.DriveRightMotor()

Ed.DriveRightMotor(direction,speed,distance)

Parameters:

Direction:

- Ed.FORWARD - Edison's right motor drives forwards.
- Ed.BACKWARD - Edison's right motor drives backwards.
- Ed.STOP - Stops Edison's right motor immediately.

Speed:

- A positive integer number between 1-10.
- Ed.SPEED_1 - PWM controlled speed setting 1.
- Ed.SPEED_2 - PWM controlled speed setting 2.
- Ed.SPEED_3 - PWM controlled speed setting 3.
- Ed.SPEED_4 - PWM controlled speed setting 4.
- Ed.SPEED_5 - PWM controlled speed setting 5.
- Ed.SPEED_6 - PWM controlled speed setting 6.
- Ed.SPEED_7 - PWM controlled speed setting 7.
- Ed.SPEED_8 - PWM controlled speed setting 8.
- Ed.SPEED_9 - PWM controlled speed setting 9.
- Ed.SPEED_10 - PWM controlled speed setting 10.
- Ed.SPEED_FULL - Full power to the motors. (Please note- Edison may not drive perfectly straight with this setting.)

Distance:

- An integer number for distance. The maximum value is 32767.

OR

- Ed.DISTANCE_UNLIMITED - turns on Edison's motors and moves on with the code. (Note: a stop will be needed later in the code.)

If using an integer number, note that the unit value of this number is set by Ed.DistanceUnits in the setup code. You can change the unit value by changing what Ed.DistanceUnits is set to in the setup code.

- Ed.DistanceUnits = Ed.CM - distance in centimetres (default for V2.0).
- Ed.DistanceUnits = Ed.INCH - distance in inches.
- Ed.DistanceUnits = Ed.TIME - distance in milliseconds (default for V1).

Returns:

N/A

Explanation:

Controls Edison's right motor to create movement. This can be set to move for a set distance (CM or INCH) or time period (TIME) and will drive the full distance before moving onto the next line of code.

Examples:

Drive Edison's right motor forward for 3 cm at speed 5. (Edison V2.0 only)

```
Ed.DistanceUnits = Ed.CM

Ed.Tempo = Ed.TEMPO_MEDIUM

#--------Your code below-----------

Ed.DriveRightMotor(Ed.FORWARD, Ed.SPEED_5, 3)
```

Drive Edison's right motor forward for 5 inches at speed 5. (Edison V2.0 only)

```
Ed.DistanceUnits = Ed.INCH

Ed.Tempo = Ed.TEMPO_MEDIUM

#--------Your code below-----------

Ed.DriveRightMotor(Ed.FORWARD, Ed.SPEED_5, 5)
```

Drive Edison's right motor forward for 2000 milliseconds at speed 7.

```
Ed.DistanceUnits = Ed.TIME

Ed.Tempo = Ed.TEMPO_MEDIUM

#--------Your code below-----------

Ed.DriveRightMotor(Ed.FORWARD, Ed.SPEED_7, 2000)
```

Set speed and distance with variables.

```
Ed.DistanceUnits = Ed.CM

Ed.Tempo = Ed.TEMPO_MEDIUM

#--------Your code below-----------

driveSpeed=5

driveDistance=10

Ed.DriveRightMotor(Ed.FORWARD, driveSpeed, driveDistance)
```

Watch out for:

Distance units of CM and INCH are only available for Edison V2.0. If you are using a V1 Edison, please make sure that Ed.DistanceUnits = Ed.TIME.

The Ed.TIME constant is in milliseconds. When Ed.DistanceUnits = Ed.TIME, remember that the distance input integer is milliseconds. Example: to drive for 2 seconds, a distance of 2000 should be input.

When distance is set to Ed.FORWARD_RIGHT, Ed.FORWARD_LEFT or Ed.SPIN_# and Ed.DistanceUnits is set to CM or INCH, any distance input will instead be considered the number of degrees that Edison will turn up to a maximum value of 360. If you put in a value above 360, the code will wrap the value around and Edison will perform a shorter turn. Example: an input of 380 will cause Edison to turn 20 degrees.

Ed.SPEED_FULL turns Edison's motors up to the maximum value. As such Edison has no control over the speed and Edison V2.0s will not be able to use their encoders to correct for driving drift.

Setting distance to 0 or Ed.DISTANCE_UNLIMITED makes Edison turn on the motors and move on with the code. An additional Ed.Drive() will be required later in the code to stop or change the direction of movement.

When the distance input is set to anything other than 0 or Ed.DISTANCE_UNLIMITED, Edison will drive for the full distance supplied before moving on to the next line of code.

Ed.ReadObstacleDetection()

Ed.ReadObstacleDetection()

Parameters:

N/A

Returns:

- Ed.OBSTACLE_NONE - Edison cannot detect an obstacle.
- Ed.OBSTACLE_RIGHT - Edison has detected an obstacle on the right.
- Ed.OBSTACLE_LEFT - Edison has detected an obstacle on the left.
- Ed.OBSTACLE_AHEAD - Edison has detected an obstacle straight ahead.

Explanation:

Reads Edison's obstacle detection state, returning its value and then clears Edison's obstacle detection register. Ed.ObstacleDetectionBeam needs to be set to ON for this function to return any value other than Ed.OBSTACLE_NONE.

Examples:

Play a beep when any obstacle is detected.

```
#--------Your code below-----------

Ed.ObstacleDetectionBeam(Ed.ON)

while True:

    if Ed.ReadObstacleDetection()>Ed.OBSTACLE_NONE:

        Ed.PlayBeep()
```

Drive until an obstacle is detected ahead.

```
#--------Your code below-----------
```

```
Ed.ObstacleDetectionBeam(Ed.ON)

Ed.Drive(Ed.FORWARD, 5, Ed.DISTANCE_UNLIMITED)

while Ed.ReadObstacleDetection() != Ed.OBSTACLE_AHEAD:

    pass

Ed.Drive(Ed.STOP, 1, 1)
```

Wait 3 seconds, then clear the obstacle detection register before looking for new obstacles to be detected and signalled with a beep.

```
#--------Your code below-----------

Ed.ObstacleDetectionBeam(Ed.ON)

Ed.TimeWait(3, Ed.TIME_SECONDS)

Ed.ReadObstacleDetection()

while Ed.ReadObstacleDetection() != Ed.OBSTACLE_AHEAD:

    pass

Ed.PlayBeep()
```

Watch out for:

Ed.ObstacleDetectionBeam needs to be set to Ed.ON for this function to return any value other than Ed.OBSTACLE_NONE.

When the obstacle detection beam is set to ON, Edison is constantly updating the obstacle detection state. This function will read the state. As such, the function may read a detection from before the read function is called in your code.

The read function clears the obstacle detection state.

When using a read function inside a loop, include a read function outside of the loop before the loop to clear any previous data.

Ed.ReadKeypad()

Ed.ReadKeypad()

Parameters:

N/A

Returns:

- Ed.KEYPAD_NONE - none of Edison's buttons have been pressed.
- Ed.KEYPAD_TRIANGLE- Edison's triangle button has been pressed.
- Ed.KEYPAD_ROUND - Edison's round button has been pressed.

Explanation:

Reads Edison's keypad state, returning its value and then clears Edison's keypad register. Edison's keypad state will be set anytime a button is pressed regardless of what the code is doing at the time.

Examples:

Play a beep when any button is pressed.

```
#--------Your code below-----------

while True:

    if Ed.ReadKeypad() != Ed.KEYPAD_NONE:

        Ed.PlayBeep()
```

Wait for the triangle button to be pressed, then beep.

```
#--------Your code below-----------

while Ed.ReadKeypad() != Ed.KEYPAD_TRIANGLE:

    pass

Ed.PlayBeep()
```

Wait 3 seconds, then clear the keypad state before looking for a new button press to be detected and signalled with a beep.

```
#--------Your code below-----------
```

```
Ed.TimeWait(3, Ed.TIME_SECONDS)

Ed.ReadKeypad()

while Ed.ReadKeypad() == Ed.KEYPAD_NONE:

    pass

Ed.PlayBeep()
```

Watch out for:

Edison is constantly updating the keypad state. This function will read the state. As such, the function may read a keypad press from before the read function is called in your code.

The read function clears the keypad state.

When using a read function inside a loop, include a read function outside of the loop before the loop to clear any previous data.

Ed.ReadClapSensor()

Ed.ReadClapSensor()

Parameters:

N/A

Returns:

- Ed.CLAP_NOT_DETECTED - Edison has not detected a clap.
- Ed.CLAP_DETECTED - Edison has detected a clap.

Explanation:

Reads Edison's clap detection state, returning its value and then clears Edison's clap detection register. Edison's clap detection state will be set anytime a clap is detected regardless of what the code is doing at the time.

Examples:

Flash an LED when a clap is detected.

```
#--------Your code below-----------

Ed.ObstacleDetectionBeam(Ed.ON)
```

```
while True:
    if ReadClapSensor()==Ed.CLAP_DETECTED:
        Ed.LeftLed(Ed.ON)
        Ed.TimeWait(50, Ed.TIME_MILLISECONDS)
        Ed.LeftLed(Ed.OFF)
        Ed.TimeWait(50, Ed.TIME_MILLISECONDS)
```

Beep when a clap is detected after a drive.

```
#--------Your code below-----------
Ed.Drive(Ed.FORWARD, 5, 10)
#wait a short time and clear the clap that was detected during the drive
Ed.TimeWait(350, Ed.TIME_MILLISECONDS)
ReadClapSensor()
#wait for a new clap
while ReadClapSensor() == Ed.CLAP_NOT_DETECTED:
    pass
Ed.PlayBeep()
```

Watch out for:

Edison is constantly updating the clap detection state. This function will read the state. As such, the function may read a clap from before the read function is called in your code.

The read function clears the clap detection state.

When using a read function inside a loop, include a read function outside of the loop before the loop to clear any previous data.

Edison's motors cause noise which will be detected and registered as "claps" while Edison is in motion. Therefore, driving Edison will cause claps to be detected. Make sure that you clear the clap detection state after waiting a few milliseconds once the driving has finished before calling the function to detect a new clap event.

Ed.ReadLineState()

Ed.ReadLineState()

Parameters:

N/A

Returns:

- Ed.LINE_ON_BLACK - Edison's line tracker is over a non-reflective surface.
- Ed.LINE_ON_WHITE- Edison's line tracker is over a reflective surface.

Explanation:

Reads the current line tracker status based on the reflected light from the line tracking sensor.

Examples:

Play a beep when a black surface is detected.

```
#--------Your code below-----------

Ed.LineTrackerLed(Ed.ON)

while True:

    if Ed.ReadLineState() == Ed.LINE_ON_BLACK:

        Ed.PlayBeep()
```

Watch out for:

Ed.LineTrackerLed() needs to be set to Ed.ON for this function to return any value.

Edison sets the LINE_ON_WHITE status when the line tracking LED is turned on and determines the LINE_ON_BLACK status by looking for a sharp drop off in reflected light. If Edison is on a black line when the line tracking LED is turned on,

Edison will mistakenly reference that value as LINE_ON_WHITE and will be unable to detect a sharp drop off to assign LINE_ON_BLACK. This will cause an error where LINE_ON_BLACK remains unset.

When the line tracker LED is ON, Edison is constantly updating the line tracker state causing this function to continuously read the current state of the line tracker.

Ed.ReadRemote()

Ed.ReadRemote()

Parameters:

N/A

Returns:

- Ed.REMOTE_CODE_NONE - Edison has not received a remote code.
- Ed.REMOTE_CODE_0 - Edison has received remote code 0.
- Ed.REMOTE_CODE_1 - Edison has received remote code 1.
- Ed.REMOTE_CODE_2 - Edison has received remote code 2.
- Ed.REMOTE_CODE_3 - Edison has received remote code 3.
- Ed.REMOTE_CODE_4 - Edison has received remote code 4.
- Ed.REMOTE_CODE_5 - Edison has received remote code 5.
- Ed.REMOTE_CODE_6 - Edison has received remote code 6.
- Ed.REMOTE_CODE_7 - Edison has received remote code 7.

Explanation:

Reads the last received remote control code and clears the remote-control register.

Examples:

Play a beep when each remote code is received in sequence.

```
#--------Your code below-----------

codes=Ed.List(8)

codes[0]=Ed.REMOTE_CODE_0

codes[1]=Ed.REMOTE_CODE_1
```

```
codes[2]=Ed.REMOTE_CODE_2

codes[3]=Ed.REMOTE_CODE_3

codes[4]=Ed.REMOTE_CODE_4

codes[5]=Ed.REMOTE_CODE_5

codes[6]=Ed.REMOTE_CODE_6

codes[7]=Ed.REMOTE_CODE_7

for x in range(8):

    while Ed.ReadRemote() != codes[x]:

        pass

    Ed.PlayBeep()

    Ed.TimeWait(500, Ed.TIME_MILLISECONDS)
```

Drive forwards while remote code 1 is being received.

```
#--------Your code below-----------

while True:

    if Ed.ReadRemote() == Ed.REMOTE_CODE_1:

        Ed.Drive(Ed.FORWARD, 5, Ed.DISTANCE_UNLIMITED)

        Ed.TimeWait(300, Ed.TIME_MILLISECONDS)

    else:

        Ed.Drive(Ed.STOP, 1, 1)
```

Watch out for:

Edison can only react to remote control codes that have been saved into the robot's memory using the barcodes provided. See https://meetedison.com/robot-activities/youre-a-robot-programmer/remote-control-barcodes/ for the full list.

Edison robots use the same IR receiver to receive remote control codes, IR data from other Edisons and perform obstacle detection. Therefore, the Ed.ObstacleDetectionBeam needs to be OFF for a remote code to be able to be received.

Edison is constantly updating the remote-control register. This function will read the state. As such, the function may read a remote-control call from before the read function is called in your code.

The read function clears the remote-control register.

When using a read function inside a loop, include a read function outside of the loop before the loop to clear any previous data.

Ed.ReadIRData()

Ed.ReadIRData()

Parameters:

N/A

Returns:

Last received infrared data from another Edison.

Explanation:

Reads the last received infrared data sent from another Edison robot.

Examples:

Play a beep when a 10 is received from another Edison.

```
#--------Your code below-----------

Ed.ReadIRData()

while True:

    if Ed.ReadIRData()==10:

        Ed.PlayBeep()

        Ed.TimeWait(500, Ed.TIME_MILLISECONDS)
```

Watch out for:

Edison robots use the same IR receiver to receive remote control codes, IR data from other Edisons and perform obstacle detection. Therefore, the Ed.ObstacleDetectionBeam needs to be OFF for data from other Edisons to be able to be received.

Edison is constantly updating the data-received register. This function will read the register. As such, the function may read data from before the read function is called in your code.

The read function clears the data-received register.

When using a read function inside a loop, include a read function outside of the loop before the loop to clear any previous data.

Ed.ReadLeftLightLevel()

Ed.ReadLeftLightLevel()

Parameters:

N/A

Returns:

The current light level of the left light sensor.

Explanation:

Reads the current light level of the left light sensor.

Examples:

Play a beep whenever the left light level reading is higher than the right light level reading.

```
#--------Your code below-----------

while True:

    if Ed.ReadLeftLightLevel()>Ed.ReadRightLightLevel():

        Ed.PlayBeep()

        Ed.TimeWait(500, Ed.TIME_MILLISECONDS)
```

Watch out for:

Edison reads the light level using an analogue to digital converter, so the returned value can be between 0 and 1023.

Ed.ReadRightLightLevel()

Ed.ReadRightLightLevel()

Parameters:

N/A

Returns:

The current light level of the right light sensor.

Explanation:

Reads the current light level of the right light sensor.

Examples:

Play a beep whenever the right light level reading is higher than the left light level reading.

```
#--------Your code below-----------

while True:

    if Ed.ReadRightLightLevel()>Ed.ReadLeftLightLevel():

        Ed.PlayBeep()

        Ed.TimeWait(500, Ed.TIME_MILLISECONDS)
```

Watch out for:

Edison reads the light level using an analogue to digital converter, so the returned value can be between 0 and 1023.

Ed.ReadLineTracker()

Ed.ReadLineTracker()

Parameters:

N/A

Returns:

The current light level of the line tracking light sensor.

Explanation:

Reads the current light level of the line tracking light sensor.

Examples:

Play a beep whenever the line tracking light level drops below a threshold buffer value.

```
#--------Your code below-----------

Ed.LineTrackerLed(Ed.ON)

white=Ed.ReadLineTracker();

buffer=150

while True:

    if Ed.ReadLineTracker()<(white-buffer):

        Ed.PlayBeep()

        Ed.TimeWait(1, Ed.TIME_SECONDS)
```

Watch out for:

Edison reads the light level using an analogue to digital converter, so the returned value can be between 0 and 1023.

This function can be used without turning the line tracking LED on, however, values returned will be much lower and harder to distinguish. Turning the line tracking LED on is highly recommended before using this read function.

Ed.ReadCountDown()

Ed.ReadCountDown(units)

Parameters:

Units:

- Ed.TIME_MILLISECONDS - Read the number of milliseconds left in the countdown timer.
- Ed.TIME_SECONDS - Read the number of seconds left in the countdown timer.

Returns:

The number of seconds or milliseconds left on the countdown timer.

Explanation:

Reads the number of seconds or milliseconds left on the countdown timer.

Examples:

Flash the left LED for 3000 milliseconds.

```
Ed.StartCountDown(3100, Ed.TIME_MILLISECONDS);

while Ed.ReadCountDown(Ed.TIME_MILLISECONDS) > 100:

    Ed.LeftLed(Ed.ON)

    Ed.TimeWait(50, Ed.TIME_MILLISECONDS)

    Ed.LeftLed(Ed.OFF)

    Ed.TimeWait(50, Ed.TIME_MILLISECONDS)
```

Watch out for:

The timer can ONLY count down, not up. It cannot be used as a stopwatch counting up, only as a timer counting down.

The timer is not a clock and cannot tell time.

When reading the timer in seconds, the read function will return an integer value, not a float value. Therefore, Ed.ReadCountDown(Ed.TIME_SECONDS) will return 0 if the time left in the countdown is less than 1 second (e.g. 0.8 seconds). Consider using milliseconds if working with values smaller than whole seconds.

Ed.ReadMusicEnd()

Ed.ReadMusicEnd()

Parameters:

N/A

Returns:

- Ed.MUSIC_FINISHED - Edison has finished playing the tune, tone or beep.
- Ed.MUSIC_NOT_FINISHED - Edison is still playing a tune, tone or beep.

Explanation:

Reads the current state of the sound being played from Edison's buzzer.

Examples:

Play a simple tune.

```
#--------Your code below-----------

simple = Ed.TuneString(25, "d4e4f4e4d4c4n2d4e4f4e4d1z")

Ed.PlayTune(simple)

while Ed.ReadMusicEnd()==Ed.MUSIC_NOT_FINISHED:

       pass
```

Watch out for:

All tunes need to end with a "z" character to end correctly.

All of Edison's sounds occur in the background, as such, Edison moves onto the next line of code as soon as the sound starts. To make Edison wait for the sound to finish use the Ed.ReadMusicEnd() function in a loop.

You can change the speed you tune plays by changing the Ed.Tempo in the setup.

Ed.ReadDriveLoad()

Ed.ReadDriveLoad()

Parameters:

N/A

Returns:

- Ed.DRIVE_NO_STRAIN - Edison's wheels are turning correctly.
- Ed.DRIVE_STRAINED - Edison's wheels are not turning.

Explanation:

Reads the current state of Edison's drive strain.

Examples:

Drive until Edison detects drive strain.

```
#--------Your code below-----------

Ed.Drive(Ed.FORWARD, 5, Ed.DISTANCE_UNLIMITED)

while Ed.ReadDriveLoad()==Ed.DRIVE_NO_STRAIN:

    pass

Ed.Drive(Ed.STOP, 1, 1)
```

Watch out for:

When driving using both wheels, detected strain on either wheel will trigger the DRIVE_STRAINED condition. When driving using only a single wheel, only strain detected on the moving wheel will trigger the DRIVE_STRAINED condition.

Ed.ReadDistance()

Ed.ReadDistance(side)

Parameters:

Side:

- Ed.MOTOR_LEFT - the number of ticks remaining on the left distance register.
- Ed.MOTOR_RIGHT- the number of ticks remaining on the right distance register.

Returns:

The number of ticks remaining on the left or right distance register.

Explanation:

Reads the number of ticks remaining on the left or right distance register.

Examples:

Drive for 40 ticks, then beep.

```
#--------Your code below-----------

Ed.Drive(Ed.FORWARD, 5, 0)

Ed.SetDistance(Ed.MOTOR_LEFT, 40)

Ed.SetDistance(Ed.MOTOR_RIGHT, 40)

while Ed.ReadDistance(Ed.MOTOR_LEFT)>0:

    pass

Ed.PlayBeep()
```

Watch out for:

This function is only compatible with Edison V2.0 robots.

The function reads values in ticks, where a tick is 1.25mm.

This read function needs to be used with the Ed.SetDistance() function.

Ed.SetDistance()

Ed.SetDistance(side,ticks)

Parameters:

Side:

- Ed.MOTOR_LEFT - set the number of ticks in the left distance register.
- Ed.MOTOR_RIGHT- set the number of ticks in the right distance register.

Ticks:

A positive integer number of ticks to set the distance register to. The maximum value is 32767. [Note: A tick is one-quarter revolution of an Edison V2.0 robot's encoder wheel and equates to 1.25mm of travel.]

Returns:

N/A

Explanation:

Sets one of the Edison V2.0 robot's distance registers, turning an unlimited drive back into a distance limited drive. Allows access to the distance registers in ticks (Edison's internal distance measurement).

Examples:

Drive for 40 ticks, then beep.

```
#--------Your code below-----------

Ed.Drive(Ed.FORWARD, 5, 0)

Ed.SetDistance(Ed.MOTOR_LEFT, 40)

Ed.SetDistance(Ed.MOTOR_RIGHT, 40)

while Ed.ReadDistance(Ed.MOTOR_LEFT)>0:

    pass

Ed.PlayBeep()
```

Watch out for:

This function is only compatible with Edison V2.0 robots.

The function reads values in ticks, where a tick is 1.25mm.

Ed.ResetDistance()

Ed.ResetDistance()

Parameters:

N/A

Returns:

N/A

Explanation:

Resets the number of ticks remaining on both the left and right distance registers to zero.

Examples:

Drive for 40 ticks or until an obstacle is encountered, then beep.

```
#--------Your code below-----------

Ed.ObstacleDetectionBeam(Ed.ON)

Ed.Drive(Ed.FORWARD, 5, 0)

Ed.SetDistance(Ed.MOTOR_LEFT, 40)

Ed.SetDistance(Ed.MOTOR_RIGHT, 40)



while Ed.ReadDistance(Ed.MOTOR_LEFT)>0:

    if Ed.ReadObstacleDetection()>Ed.OBSTACLE_NONE:

        Ed.ResetDistance()

Ed.PlayBeep()
```

Watch out for:

This function is only compatible with Edison V2.0 robots.

The function reads values in ticks, where a tick is 1.25mm.

Ed.ON

Ed.ON

Value:

Ed.ON = 1

Used in:

- Ed.LeftLed()
- Ed.RightLed()
- Ed.ObstacleDetectionBeam()
- Ed.LineTrackerLed()

Ed.OFF

Ed.OFF

Value:

Ed.OFF = 0

Used in:

- Ed.LeftLed()
- Ed.RightLed()
- Ed.ObstacleDetectionBeam()
- Ed.LineTrackerLed()

Ed.NOTE_#

Ed.NOTE_#

Value:

Ed.NOTE_A_6	=	18181
Ed.NOTE_A_SHARP_6	=	17167
Ed.NOTE_B_6	=	16202

Ed.NOTE_C_7	=	15289
Ed.NOTE_C_SHARP_7	=	14433
Ed.NOTE_D_7	=	13622
Ed.NOTE_D_SHARP_7	=	12856
Ed.NOTE_E_7	=	12135
Ed.NOTE_F_7	=	11457
Ed.NOTE_F_SHARP_7	=	10814
Ed.NOTE_G_7	=	10207
Ed.NOTE_G_SHARP_7	=	9632
Ed.NOTE_A_7	=	9090
Ed.NOTE_A_SHARP_7	=	8581
Ed.NOTE_B_7	=	8099
Ed.NOTE_C_8	=	7644
Ed.NOTE_REST	=	0
Ed.NOTE_SIXTEENTH	=	125
Ed.NOTE_EIGHTH	=	250
Ed.NOTE_QUARTER	=	500
Ed.NOTE_HALF	=	1000
Ed.NOTE_WHOLE	=	2000

Used in:

- Ed.PlayTone
  - Ed.NOTE_A_6 - Play a low A.
  - Ed.NOTE_A_SHARP_6 - Play a low A sharp.
  - Ed.NOTE_B_6 - Play a low B.
  - Ed.NOTE_C_7 - Play a C.
  - Ed.NOTE_C_SHARP_7 - Play a C sharp.
  - Ed.NOTE_D_7 - Play a D.
  - Ed.NOTE_D_SHARP_7 - Play a D sharp.
  - Ed.NOTE_E_7 - Play an E.
  - Ed.NOTE_F_7 - Play an F.
  - Ed.NOTE_F_SHARP_7 - Play an F sharp.
  - Ed.NOTE_G_7 - Play a G.
  - Ed.NOTE_G_SHARP_7 - Play a G sharp.
  - Ed.NOTE_A_7 - Play an A.
  - Ed.NOTE_A_SHARP_7 - Play an A sharp.
  - Ed.NOTE_B_7 - Play a B.
  - Ed.NOTE_C_8 - Play a high C.
  - Ed.NOTE_REST - Play a rest.
  - Ed.NOTE_SIXTEENTH - Play note for 125 milliseconds.
  - Ed.NOTE_EIGHTH - Play note for 250 milliseconds.
  - Ed.NOTE_QUARTER - Play note for 500 milliseconds.
  - Ed.NOTE_HALF - Play note for 1000 milliseconds.
  - Ed.NOTE_WHOLE - Play note for 2000 milliseconds.

Ed.TEMPO_#

Ed.TEMPO_#

Value:

Ed.TEMPO_VERY_SLOW	=	1000
Ed.TEMPO_SLOW	=	500
Ed.TEMPO_MEDIUM	=	250
Ed.TEMPO_FAST	=	70
Ed.TEMPO_VERY_FAST	=	1

Used with:

- Ed.Tempo

Ed.FORWARD

Ed.FORWARD

Value:

Ed.FORWARD = 1

Used in:

- Ed.Drive()
- Ed.DriveLeft()
- Ed.DriveRight()

Ed.BACKWARD

Ed.BACKWARD

Value:

Ed.BACKWARD = 2

Used in:

- Ed.Drive()
- Ed.DriveLeft()
- Ed.DriveRight()

Ed.FORWARD_RIGHT

Ed.FORWARD_RIGHT

Value:

Ed.FORWARD_RIGHT = 3

Used in:

- Ed.Drive()

Ed.BACKWARD_RIGHT

Ed.BACKWARD_RIGHT

Value:

Ed.BACKWARD_RIGHT = 4

Used in:

- Ed.Drive()

Ed.FORWARD_LEFT

Ed.FORWARD_LEFT

Value:

Ed.FORWARD_LEFT = 5

Used in:

- Ed.Drive()

Ed.BACKWARD_LEFT

Ed.BACKWARD_LEFT

Value:

Ed.BACKWARD_LEFT = 6

Used in:

- Ed.Drive()

Ed.SPIN_#

ED.SPIN_#

Value:

Ed.SPIN_RIGHT	=	7
Ed.SPIN_LEFT	=	8

Used in:

- Ed.Drive()

Ed.STOP

Ed.STOP

Value:

Ed.STOP = 0

Used in:

- Ed.Drive()
- Ed.DriveLeft()
- Ed.DriveRight()

Ed.SPEED_#

Ed.SPEED_#

Value:

Ed.SPEED_1	=	1
Ed.SPEED_2	=	2
Ed.SPEED_3	=	3
Ed.SPEED_4	=	4
Ed.SPEED_5	=	5
Ed.SPEED_6	=	6
Ed.SPEED_7	=	7
Ed.SPEED_8	=	8
Ed.SPEED_9	=	9
Ed.SPEED_10	=	10
Ed.SPEED_FULL	=	0

Used in:

- Ed.Drive()
- Ed.DriveLeft()
- Ed.DriveRight()

Ed.DISTANCE_UNLIMITED

Ed.DISTANCE_UNLIMITED

Value:

Ed.DISTANCE_UNLIMITED = 0

Used in:

- Ed.Drive()
- Ed.DriveLeft()
- Ed.DriveRight()

Ed.MOTOR_#

Ed.MOTOR_#

Value:

MOTOR_LEFT	=	0
MOTOR_RIGHT	=	1

Used in:

- Ed.ReadDistance()
- Ed.SetDistance()

Ed.TIME_#

Ed.TIME_#

Value:

TIME_SECONDS	=	0
TIME_MILLISECONDS	=	1

Used in:

- Ed.StartCountDown()
- Ed.TimeWait()
- Ed.ReadCountDown()

Ed.OBSTACLE_#

Ed.OBSTACLE_#

Value:

OBSTACLE_NONE	=	0
OBSTACLE_RIGHT	=	0x08
OBSTACLE_LEFT	=	0x20
OBSTACLE_AHEAD	=	0x10

Used with:

- Ed.ReadObstacleDetection()

Ed.LINE_ON_#

Ed.LINE_ON_#

Value:

LINE_ON_BLACK	=	1
LINE_ON_WHITE	=	0

Used with:

- Ed.ReadLineState()

Ed.KEYPAD_#

Ed.KEYPAD_#

Value:

KEYPAD_NONE	=	0
KEYPAD_TRIANGE	=	1
KEYPAD_ROUND	=	4

Used with:

- Ed.ReadKeypad()

Ed.CLAP_#

Ed.CLAP_#

Value:

CLAP_NOT_DETECTED	=	0
CLAP_DETECTED	=	4

Used in:

- Ed.ReadClapSensor()

Ed.DRIVE_#

Ed.DRIVE_#

Value:

DRIVE_STRAINED	=	1
DRIVE_NO_STRAIN	=	0

Used with:

- Ed.ReadDriveLoad()

Ed.MUSIC_#

Ed.MUSIC_#

Value:

MUSIC_FINISHED	=	1
MUSIC_NOT_FINISHED	=	0

Used with:

- Ed.ReadMusicEnd()

Ed.REMOTE_CODE_#

Ed.REMOTE_CODE_#

Value:

REMOTE_CODE_1	=	1
REMOTE_CODE_2	=	2
REMOTE_CODE_3	=	3
REMOTE_CODE_4	=	4
REMOTE_CODE_5	=	5
REMOTE_CODE_6	=	6

REMOTE_CODE_7	=	7
REMOTE_CODE_NONE	=	255

Used with:

- Ed.ReadRemote()

Ed.EVENT_#

Ed.EVENT_#

Value:

EVENT_TIMER_FINISHED	=	0
EVENT_REMOTE_CODE	=	1
EVENT_IR_DATA	=	2
EVENT_CLAP_DETECTED	=	3
EVENT_OBSTACLE_ANY	=	4
EVENT_OBSTACLE_LEFT	=	5
EVENT_OBSTACLE_RIGHT	=	6
EVENT_OBSTACLE_AHEAD	=	7
EVENT_DRIVE_STRAIN	=	8
EVENT_KEYPAD_TRIANGLE	=	9
EVENT_KEYPAD_ROUND	=	10
EVENT_LINE_TRACKER_ON_WHITE	=	11
EVENT_LINE_TRACKER_ON_BLACK	=	12

EVENT_LINE_TRACKER_SURFACE_CHANGE	=	13
EVENT_TUNE_FINISHED	=	14

Used in:

- Ed.RegisterEventHandler()
  - Ed.EVENT_TIMER_FINISHED - Calls the function when the countdown timer finishes.
  - Ed.EVENT_REMOTE_CODE - Calls the function when Edison receives a remote code.
  - Ed.EVENT_IR_DATA - Calls the function when Edison receives code from another Edison.
  - Ed.EVENT_CLAP_DETECTED - Calls the function when Edison detects a clap.
  - Ed.EVENT_OBSTACLE_ANY - Calls the function when Edison detects any obstacle.
  - Ed.EVENT_OBSTACLE_LEFT - Calls the function when Edison detects an obstacle to the left.
  - Ed.EVENT_OBSTACLE_RIGHT - Calls the function when Edison detects an obstacle to the right.
  - Ed.EVENT_OBSTACLE_AHEAD - Calls the function when Edison detects an obstacle straight ahead.
  - Ed.EVENT_DRIVE_STRAIN - Calls the function when Edison detects strain on the drive.
  - Ed.EVENT_KEYPAD_TRIANGLE - Calls the function when Edison detects a triangle button press.
  - Ed.EVENT_KEYPAD_ROUND - Calls the function when Edison detects a round button press.
  - Ed.EVENT_LINE_TRACKER_ON_WHITE - Calls the function when Edison detects a white surface under the line tracker.
  - Ed.EVENT_LINE_TRACKER_ON_BLACK - Calls the function when Edison detects a black surface under the line tracker.
  - Ed.EVENT_LINE_TRACKER_SURFACE_CHANGE - Calls the function when Edison detects a surface change under the line tracker.
  - Ed.EVENT_TUNE_FINISHED - Calls the function when Edison finishes playing a tune.

Ed.CM

Ed.CM

Value:

Ed.CM = 0

Used with:

- Ed.DistanceUnits
  - Sets units to CM.

Ed.INCH

Ed.INCH

Value:

Ed.INCH = 1

Used with:

- Ed.DistanceUnits
  - Sets units to inches.

Ed.TIME

Ed.TIME

Value:

Ed.TIME = 2

Used with:

- Ed.DistanceUnits
  - Sets units to milliseconds.

Ed.V1

Ed.V1

Value:

Ed.V1 = 1

Used with:

- Ed.EdisonVersion
  - Makes the code compile for Edison version 1 robots.

Ed.V2

Ed.V2

Value:

Ed.V2 = 2

Used with:

- Ed.EdisonVersion
  - Makes the code compile for Edison version 2 robots.

Ed.Tempo

Ed.Tempo

Can be set to:

- Ed.TEMPO_VERY_SLOW - Play tunes very slowly.
- Ed.TEMPO_SLOW - Play tunes slowly.
- Ed.TEMPO_MEDIUM - Play tunes normally.
- Ed.TEMPO_FAST - Play tunes fast.
- Ed.TEMPO_VERY_FAST- Play tunes very fast.

Explanation:

Changes how fast or slow Edison plays a tune.

Examples:

Play a simple tune very fast.

```
Ed.Tempo = Ed.TEMPO_VERY_FAST

#--------Your code below-----------

simple = Ed.TuneString(25, "d4e4f4e4d4c4n2d4e4f4e4d1z")

Ed.PlayTune(simple)

while Ed.ReadMusicEnd()==Ed.MUSIC_NOT_FINISHED:

    pass
```

Watch out for:

Ed.Tempo has to be set to one of the constants specified above.

Ed.Tempo can only be set once per program and must be set in the 'Setup' code.

Ed.Tempo is set to Ed.TEMPO_MEDIUM by default. This can be changed in the 'Setup' code section.

Ed.DistanceUnits

Ed.DistanceUnits

Can be set to:

- Ed.CM - Sets drive distances units to cm.
- Ed.INCH - Sets drive distances units to inches.
- Ed.TIME - Sets drive distances units to milliseconds.

Explanation:

Changes the units of the distance used in all drive functions in a program.

Examples:

Drive Edison forward for 3 cm at speed 5. (Edison V2.0 only)

```
Ed.DistanceUnits = Ed.CM
```

```
Ed.Tempo = Ed.TEMPO_MEDIUM

#--------Your code below-----------

Ed.Drive(Ed.FORWARD, Ed.SPEED_5, 3)
```

Drive Edison forward for 5 inches at speed 5. (Edison V2.0 only)

```
Ed.DistanceUnits = Ed.INCH

Ed.Tempo = Ed.TEMPO_MEDIUM

#--------Your code below-----------

Ed.Drive(Ed.FORWARD, Ed.SPEED_5, 5)
```

Drive Edison forward for 2000 milliseconds at speed 7.

```
Ed.DistanceUnits = Ed.TIME

Ed.Tempo = Ed.TEMPO_MEDIUM

#--------Your code below-----------

Ed.Drive(Ed.FORWARD, Ed.SPEED_7, 2000)
```

Watch out for:

Ed.DistanceUnits has to be set to one of the constants specified above.

Ed.DistanceUnits can only be set once per program and must be set in the 'Setup' code.

Ed.DistanceUnits is set to Ed.CM by default. This can be changed in the 'Setup' code section.

Version 1 Edison robots can only use Ed.DistanceUnits = Ed.TIME.

Ed.EdisonVersion

Ed.EdisonVersion

Can be set to:

- Ed.V1 - Version 1 Edison robot.
- Ed.V2 - Version 2.0 Edison robot.

Explanation:

Sets the version of Edison for which the code is compiled. Version 2.0 Edison robots contain encoders, allowing them to drive for very precise distances.

Version 2.0 Edison robots can be set to drive in cm or inches and can read and write to the distance counter register.

Examples:

Edison Version 1 set up.

```
#-------------Setup----------------

import Ed

Ed.EdisonVersion = Ed.V1

Ed.DistanceUnits = Ed.TIME

Ed.Tempo = Ed.TEMPO_MEDIUM

#--------Your code below-----------
```

Edison Version 2.0 set up.

```
#-------------Setup----------------

import Ed

Ed.EdisonVersion = Ed.V2

Ed.DistanceUnits = Ed.CM
```

```
Ed.Tempo = Ed.TEMPO_MEDIUM

#--------Your code below-----------
```

Watch out for:

Ed.EdisonVersion has to be set to one of the constants specified above.

Ed.EdisonVersion can only be set once per program and must be set in the 'Setup' code.

Ed.EdisonVersion is set using the version pop up box which opens when EdPy launches by default. This can be changed in the 'Setup' code section.

Version 1 Edison robots can only use Ed.DistanceUnits = Ed.TIME.

abs()

abs(int)

Parameters:

Int:

An integer value between +/- 32767

Returns:

The mathematical absolute value of the input.

Explanation:

Flips the sign of any negative number input, returning a positive number without changing the absolute value.

Examples:

Change negative 30 to positive 30 and check by beeping.

```
#--------Your code below-----------

x=-30

x=abs(x)

if x==30:
```

```
    Ed.PlayBeep()
```

Watch out for:

abs() is a native python function. As such, it does not need the "Ed." prefix.

len()

len(list)

Parameters:

List:

An EdPy list.

Returns:

The number of elements in the EdPy list.

Explanation:

Returns the maximum number of integers that can be stored in the given list.

Examples:

Loop through a created list and beep when a 2 is found.

```
#--------Your code below-----------

exampleList = Ed.List(6,[1,2,3,3,2,1])

listSize=len(exampleList)

for x in range(listSize):

    if exampleList[x]==2:

        Ed.PlayBeep()

        Ed.TimeWait(500, Ed.TIME_MILLISECONDS)
```

Watch out for:

len() returns the value you entered as the size of the list when it was created, even if you have not assigned values to all of the integers.

range()

range(limit)

Parameters:

Limit:

The value to count up to.

Returns:

The values from 0 to the limit in sequence.

Explanation:

Controls the number of times the code loops around. Only valid in "for" loops.

Produces values of 0 to limit minus 1 (i.e. limit - 1) for a “for” loop.

Examples:

Loop through a created list and beep when a 2 is found.

```
#--------Your code below-----------

exampleList = Ed.List(6,[1,2,3,3,2,1])

listSize=len(exampleList)

for x in range(listSize):

    if exampleList[x]==2:

        Ed.PlayBeep()

        Ed.TimeWait(500, Ed.TIME_MILLISECONDS)
```

Watch out for:

Limit makes a for loop count up from zero, up to one value under the limit. The for loop does not use the number entered as the limit.

True

True

Value:

True = 1

Used with:

- while
- if

Explanation:

Passes a success value to a while or if test.

Example:

Loop forever flashing an LED.

```
#--------Your code below-----------

while True:

    Ed.RightLed(Ed.ON)

    Ed.TimeWait(50, Ed.TIME_MILLISECONDS)

    Ed.RightLed(Ed.OFF)

    Ed.TimeWait(50, Ed.TIME_MILLISECONDS)
```

False

False

Value:

False = 0

Used with:

- while
- if

Explanation:

Passes a failure value to a while or if test.

Example:

Skip a loop.

```
#--------Your code below-----------

while False:

    Ed.RightLed(Ed.ON)

    Ed.TimeWait(50, Ed.TIME_MILLISECONDS)

    Ed.RightLed(Ed.OFF)

    Ed.TimeWait(50, Ed.TIME_MILLISECONDS)
```

import

import

Can be imported:

- Ed

Explanation:

Imports the Ed. library to make EdPy compatible with Edison.

def

def

Explanation:

Used to define a new user function. A function can be defined before or after the main body of the code and used anywhere in the main body of the code.

User functions can be defined to take input values and return values back to the main code.

Examples:

A function that waits for an obstacle to be detected ahead of Edison.

```
#--------Your code below-----------

Ed.ObstacleDetectionBeam(Ed.ON)

waitObs()

Ed.PlayBeep()

def waitObs():

    while Ed.ReadObstacleDetection() != Ed.OBSTACLE_AHEAD:

        Ed.TimeWait(50, Ed.TIME_MILLISECONDS)
```

A function that takes in an input and returns a value based on the input.

```
#--------Your code below-----------

for x in range(10):

    if isTwo(x):

        Ed.PlayBeep()

    else:

        Ed.LeftLed(Ed.ON)



    Ed.TimeWait(500, Ed.TIME_MILLISECONDS)

    Ed.LeftLed(Ed.OFF)

    Ed.TimeWait(500, Ed.TIME_MILLISECONDS)

def isTwo(number):
```

```
    if number==2:

        return True

    else:

        return False
```

pass

pass

Used with:

- if
- while
- for

Explanation:

Used to do nothing in an if statement or loop.

Examples:

Flash the right LED for every loop except the 3rd.

```
#--------Your code below-----------

for x in range(10):

    if x==3:

        pass

    else:

        Ed.RightLed(Ed.ON)

        Ed.TimeWait(50, Ed.TIME_MILLISECONDS)

        Ed.RightLed(Ed.OFF)
```

```
        Ed.TimeWait(50, Ed.TIME_MILLISECONDS)
```

Keep looping until a key is pressed. (I.e. wait until a key is pressed.)

```
#--------Your code below-----------

while Ed.ReadKeypad() ==Ed.KEYPAD_NONE:

        pass

Ed.PlayBeep()
```

for

for

Explanation:

Used with the range function to loop indented code for a set number of times.

Examples:

Flash the right LED 10 times.

```
#--------Your code below-----------

for x in range(10):

    Ed.RightLed(Ed.ON)

    Ed.TimeWait(100, Ed.TIME_MILLISECONDS)

    Ed.RightLed(Ed.OFF)

    Ed.TimeWait(100, Ed.TIME_MILLISECONDS)
```

Set up a list with all elements set to zero.

```
#--------Your code below-----------
```

```
zeros=Ed.List(5)

for x in range(5):

    zeros[x]=0
```

while

while

Explanation:

Used to repeat indented code until the condition following the while evaluates to false.

Examples:

Flash the right LED 10 times.

```
#--------Your code below-----------

x=1

while x != 10:

    Ed.RightLed(Ed.ON)

    Ed.TimeWait(100, Ed.TIME_MILLISECONDS)

    Ed.RightLed(Ed.OFF)

    Ed.TimeWait(100, Ed.TIME_MILLISECONDS)

    x=x+1
```

Loop until an obstacle is detected in front of Edison.

```
#--------Your code below-----------

while Ed.ReadObstacleDetection() != Ed.OBSTACLE_AHEAD:

    pass
```

```
Ed.PlayBeep()
```

if/elif/else

if

Explanation:

Used to run indented code if the condition following the if evaluates to true.

else

Explanation:

Used to run indented code if the condition following the if evaluates to false.

elif

Explanation:

Used to run indented code if the condition following the if evaluates to false and the elif evaluates to true.

Examples:

Loop 10 times flashing each of the times other than the 5th time where Edison beeps instead.

```
#--------Your code below-----------

testVal=1

while True:

    if testVal==11:

        break

    elif testVal==5:

            Ed.PlayBeep()

            Ed.TimeWait(1000, Ed.TIME_MILLISECONDS)
```

```
    else:
        Ed.LeftLed(Ed.ON)
        Ed.TimeWait(500, Ed.TIME_MILLISECONDS)
        Ed.LeftLed(Ed.OFF)
        Ed.TimeWait(500, Ed.TIME_MILLISECONDS)
    testVal=testVal+1
```

class

class

Explanation:

Used to define a new type of Python object.

A python object is a collection of variables and functions that act on those variables. Multiple objects can be created from a single class definition and changing the values in one created object does not change the values in any other created object.

Examples:

Create a single object.

```
x=notes()
Ed.PlayTone(x.i,x.b)
while Ed.ReadMusicEnd() == Ed.MUSIC_NOT_FINISHED:
    pass
class notes:
    def_init_(self):
        self.i=1000
```

```
        self.b=2000

    def changeB(self,newB):

        self.b=newB

    def changeI(self,newI):

        self.i=newI
```

Create two objects based on a single class and change the values in one of the objects.

```
#----------------Your code below-------------------------

x=notes()

y=notes()

y.changeB(3000)

y.changeI(32000)



Ed.PlayTone(x.i,x.b)

while Ed.ReadMusicEnd() == Ed.MUSIC_NOT_FINISHED:

    pass

Ed.TimeWait(10, Ed.TIME_MILLISECONDS)

Ed.PlayTone(y.i,y.b)



class notes:

    def __init__(self):

        self.i=1000

        self.b=2000
```

```
    def changeB(self,newB):
        self.b=newB


    def changeI(self,newI):
        self.i=newI
```

Watch out for:

Functions defined in a class must have "self" as the first argument in the definition but this argument is ignored when calling the function from the main code.

All classes need a function called _init_() which is referred to as a creator and is called when a new object is created. The _init_() function is used to set the initial values of the variables in the object. The _init_() function can be defined with inputs which need to be set as the object is created. See example:

```
x=notes(1000,2000)
Ed.PlayTone(x.i,x.b)
while Ed.ReadMusicEnd() == Ed.MUSIC_NOT_FINISHED:
    pass
class notes:
    def_init_(self,newI,newB):
        self.i=newI
        self.b=newB
    def changeB(self,newB):
        self.b=newB


    def changeI(self,newI):
```

```
        self.i=newI
```

return

return

Explanation:

Used to return a value from a function.

The returned value must be an integer.

Examples:

A function that takes in an input and returns a value based on the input.

```
#--------Your code below-----------

for x in range(10):

    if isTwo(x):

        Ed.PlayBeep()

    else:

        Ed.LeftLed(Ed.ON)

    Ed.TimeWait(500, Ed.TIME_MILLISECONDS)

    Ed.LeftLed(Ed.OFF)

    Ed.TimeWait(500, Ed.TIME_MILLISECONDS)

def isTwo(number):

    if number==2:

        return True

    else:

        return False
```