



2025.10.31

RobomationLAB

코딩 가이드



Robomation

개요

이 문서는 (주)로보메이션에서 개발한 '로봇 동작 스트리밍 서비스를 위한 실행 엔진 및 통합 저작환경' 중 하나인 RobomationLAB(로보메이션 랩) 에서 사용되는 로봇 소프트웨어 모델링 원리와 기본 코딩 문법 및 함수들을 설명합니다.

목차

1. RobomationLAB 로봇 코딩 교육 플랫폼
 - 1-1) 로봇 코딩 프로그램
 - 1-2) 코딩 프로그램 주요 특징
 - 1-3) 실시간 로봇 제어 방식
2. RobomationLAB 로봇 프로그래밍 방식
 - 2-1) 순차 실행과 병렬 실행
 - 2-2) setup 함수
 - 2-3) loop 함수
3. 로봇 소프트웨어 모델링
 - 3-1) 모델링 과정
 - 3-2) Roboid 클래스 정의
 - 3-3) 데이터타입(DataType)
 - 3-4) 고유 식별자, urn
4. RobomationLAB 로봇 코딩 기본 문법 체계
 - 4-1) 로봇 Device 객체
 - 4-2) d 함수
 - 4-3) c 함수
 - 4-4) e 함수
 - 4-5) w 함수
5. RobomationLAB 커스텀 함수
 - 5-1) 기본 유틸리티 함수
 - 5-2) 로봇 별 특수 기능 함수

RobomationLAB 로봇 코딩 교육 플랫폼

RobomationLAB 은 초등/중학생 대상 로봇 코딩 교육을 위한 웹 브라우저 기반의 통합 저작환경을 제공합니다. 로봇 하드웨어를 추상화해 소프트웨어로써 모델링한 '로보이드 모델'을 활용하여, 코드 해석 및 실행을 통해 자사 로봇들을 실시간으로 제어할 수 있는 실행 환경을 제공합니다.

로봇 코딩 프로그램

RobomationLAB에서 제공하는 로봇 코딩 프로그램은 다음과 같습니다.

1. Block Composer (블록 컴포저)

Block Composer는 블록 코딩을 통해 자사 로봇들을 쉽고 빠르게 제어하며, 로봇 제어의 기초를 학습할 수 있는 도구입니다.

- 피지컬 컴퓨팅에 최적화된 저작 환경
- 블록 Drag & Drop 방식으로 초보자도 쉽게 코딩 가능
- 기본 개념부터 문법 오류 없는 학습 환경 제공
- Javascript, Python 스크립트 코드로 자동 변환
- 로봇 별로 미리 정해진 기능을 가진 블록 모음과 다양한 체험 예제 제공
- 코드 실행을 통해 실시간 결과 확인 가능
- 블록 조합으로 문제 해결 능력 및 창의력 향상
- AI 기반 스크립트 코드 분석을 통해 최적화된 피드백 제공

2. Script Composer (스크립트 컴포저)

Script Composer는 스크립트 코딩을 통해 자사 로봇들을 쉽고 빠르게 제어하며, Javascript, Python 문법 및 로봇 코딩의 기초를 학습할 수 있는 도구입니다.

- Javascript, Python 에디터 동시 제공
- 언어 별 코드 자동 완성 및 코드 삽입 기능 제공
- 로봇 별 다양한 체험 예제 코드 제공
- 코드 실행을 통해 실시간 결과 확인 가능
- AI 기반 스크립트 코드 분석을 통해 최적화된 피드백 제공

코딩 프로그램 주요 특징

RobomationLAB 에서 제공하는 로봇 코딩 프로그램의 주요 특징은 다음과 같습니다.

1. 크롬 웹 브라우저 기반으로, OS(운영체제)의 제약을 받지 않음
2. Web Serial 통신 기반으로, USB 동글을 통해 로봇 하드웨어를 직접 제어
3. 멀티 로봇 동시 제어 지원 - 로봇의 종류, 수 제한 없음
4. 파일 저장 시 결과물은 JSON 텍스트 파일로 변환하여 저장

실시간 로봇 제어 방식

RobomationLAB에서 제공하는 로봇 코딩 프로그램에서는 다음과 같은 과정을 통해 실시간으로 로봇을 제어합니다.

1. 블록코딩 또는 스크립트 코딩을 통해,
로봇을 제어하기 위한 Effector, Command 객체의 값을 설정하거나
로봇의 Sensor 값 및 Event 발생을 활용하는 코드를 작성합니다.
2. 코드를 실행합니다.
3. Web Serial 통신을 통해 로봇으로부터 Sensor 및 Event 데이터를 담은 패킷을 받아,
로봇 Device 객체에 반영합니다.
4. 실시간으로 코드를 해석하여,
Effector, Command 객체에 데이터를 덮어쓰거나, Sensor, Event 객체의 값을 읽어옵니다.
5. 로봇 Device 객체의 데이터를 반영한 패킷을 생성한 뒤,
Web Serial 통신을 통해 로봇에 전송하여 실제로 로봇이 동작하는지 확인합니다.
6. 코드가 실행되는 동안, 3 ~ 5 의 과정을 약 10 ~ 20ms 마다 반복 수행합니다.

RobomationLAB 로봇 프로그래밍 방식

순차 실행과 병렬 실행

로봇을 프로그래밍하는 방식에는, 순차 실행 방식과 병렬 실행 방식이 있습니다.

순차 실행은, 한 동작이 마무리된 뒤에 다음 동작을 수행하는 방식으로, 단순한 행동을 코딩하는 데에 적합합니다. 예를 들어, 로봇을 앞으로 이동한 이후에 멈춰서 LED를 켜고 싶다면, 각 동작에 해당하는 코드를 순서대로 배치하여 시간순으로 실행할 수 있도록 순차 실행 방식이 가능해야 합니다.

병렬 실행은, 여러 동작을 동시에 수행하는 방식으로, 보다 복잡하고 고도의 행동을 프로그래밍하는 데에 필요합니다. 예를 들어, 2족 보행 로봇이 걷는 동작을 구현하고자 한다면, 로봇의 발과 다리들을 동시에 움직여야 보행이 가능하기 때문에 병렬 실행 방식의 코딩이 가능해야 합니다.

RobomationLAB에서 제공하는 로봇 코딩 프로그램은, Arduino의 H/W 개발 환경과 유사한 setup / loop 코드 구조를 기반으로, 순차 실행 방식과 병렬 실행 방식을 동시에 지원합니다.



Block Composer 에 처음 접속하면, 다음과 같이 두 개의 빈 함수 블록이 작업공간에 표시되는데, '시작하기' 블록은 setup 함수를, '무한 반복하기' 블록은 loop 함수를 나타냅니다.

언어에 따라 다음과 같은 기본 코드 구조를 갖습니다.

- 자바스크립트

```
// put setup code here, to run once:
async function setup() {
}

// put control code here, to run repeatedly:
function loop() {
}
```

- 파이썬

```
import asyncio

# put setup code here, to run once:
async def setup():
    pass
```

```
# put control code here, to run repeatedly:
def loop():
    pass
```

setup 함수

setup 함수는 '코드 실행'을 하는 순간 단 한번만 수행됩니다.

setup 함수에서는 주로 변수를 초기화하거나 로봇의 모드, 기능 등을 초기화하는 코드를 작성합니다.

예를 들어, 바퀴를 통해 움직이는 로봇을 제어할 때, setup 함수에서는 바퀴의 초기 속도를 설정할 수 있습니다.

또한, setup 함수 앞에 붙은 async 키워드는, 함수 내의 코드를 순차 실행 할 수 있도록 지원합니다.

async/await 방식은 최신 자바스크립트 및 파이썬에 추가된 방식으로, 특정 함수를 비동기 함수로 지정하여, 다른 코드의 실행에 영향을 주지 않으면서 해당 함수의 동작이 완료될 때까지 기다립니다.

실제로 setup 함수는 코드 실행 초기에 한번만 수행되지만, 함수 내에 await 키워드가 붙은 코드가 있는 경우 정해진 시간 또는 동작 이후에 깨어나 다음 코드를 수행하기 때문에, 계속 활성화 되어 있다고 볼 수 있습니다. 이 기능을 활용하면, 간단한 순차 실행 뿐 아니라 병렬 실행 역할을 하는 loop 함수와의 연동을 통해, 강력한 로봇 프로그래밍이 가능할 것입니다.

다음은 HamsterS 로봇이 1초간 전진 후 1초간 후진하는 코드를 작성하는 예시입니다.

병렬 실행 방식의 loop 함수 내에서 위 동작을 구현하려면, 시간 계산과 제어 코드가 혼합되어 코드가 매우 복잡해지게 됩니다. 대신, setup 함수 내에서 await 키워드가 붙은 '_wait' 시간 지연 함수를 사용하면, 마치 동기 방식처럼 시간순으로 동작하는 코드를 작성할 수 있습니다.

(_wait 함수에 대해서는 이후 '기본 유틸리티 함수' 에서 다시 설명합니다.)

- 예시 코드 (자바스크립트)

```
// put setup code here, to run once:
async function setup() {
    // 양쪽 바퀴 속도를 50으로 설정하여 앞으로 이동
    $('HamsterS*0:wheel.speed.left').d = 50;
    $('HamsterS*0:wheel.speed.right').d = 50;
    await _wait(1000); // 1초 (1000밀리초) 기다리기
    // 양쪽 바퀴 속도를 -50으로 설정하여 뒤로 이동
```

```

    $('HamsterS*0:wheel.speed.left').d = -50;
    $('HamsterS*0:wheel.speed.right').d = -50;
    await __wait(1000); // 1초 (1000밀리초) 기다리기
  }
  // put control code here, to run repeatedly:
  function loop() {
  }

```

(\$ 문법에 대해서는 이후 'RobomationLAB 로봇 코딩 기본 문법' 에서 다시 설명합니다.)

loop 함수

loop 함수는 병렬 실행을 지원하며, 코드가 실행되는 동안 약 10 ~ 20ms 마다 반복하여 수행됩니다.

loop 함수에서는 주로 변수의 값을 반복해서 설정하거나 로봇의 특정 이벤트의 발생을 감지하여 처리하는 코드를 작성합니다.

다음은 시간에 따라 HamsterS 로봇의 바퀴 속도와 LED 밝기가 변하는 코드를 작성하는 예시입니다.

- 예시 코드 (자바스크립트)

```

var frame;
// put setup code here, to run once:
async function setup() {
  frame = 0;
}
// put control code here, to run repeatedly:
function loop() {
  frame += 1; // loop 함수가 호출될 때마다 frame 변수의 값 1씩 증가

  // 바뀐 frame 값을 활용하여, 양쪽 바퀴 속도와 양쪽 LED의 RGB 값 설정
  $('HamsterS*0:wheel.speed.left').d = frame % 100;
  $('HamsterS*0:wheel.speed.right').d = frame % 100;
  $('HamsterS*0:led.left').d = [frame % 256, 0, 0];
  $('HamsterS*0:led.right').d = [0, 0, frame % 256];
}

```

다음은 HamsterS 로봇의 몸체를 가볍게 두드리는 Tap 동작이 발생하면, LED를 적색으로 켜는 코드를 작성하는 예시입니다.

- 예시 코드 (파이썬)

```
import asyncio

# put setup code here, to run once:
async def setup():
    pass

# put control code here, to run repeatedly:
def loop():
    # Tap 동작이 발생하는 순간, 이벤트 발생 감지
    if __('HamsterS*0:gravity.tap').e == True: # 이벤트 감지 시, True가 됨
        __('HamsterS*0:led.left').d = [255, 0, 0] # LED의 Red를 255로 설정
    else:
        __('HamsterS*0:led.left').d = [0, 0, 0]
    return
```

(__ 와 .e 문법 에 대해서는 이후 'RobomationLAB 로봇 코딩 기본 문법' 에서 다시 설명합니다.)

로봇 소프트웨어 모델링

스트리밍 방식의 로봇 하드웨어 제어를 위한 최소 시스템 구성과 JSON 기반의 마크업 문법을 정의합니다. 블록 Drag & Drop 방식의 GUI 모델러인 Roboid Modeler 를 활용하여 로봇 소프트웨어 모델링을 수행합니다.

모델링 과정

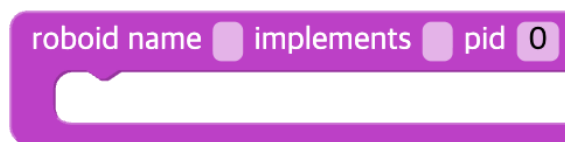
로봇 소프트웨어 모델링 과정은 크게 다음과 같습니다.

1. 로봇 하드웨어가 가지고 있는 기능, 센서, 특징 등을 추상화하여 정리합니다.
2. 각각 클래스, 이름, 데이터 타입 등이 사전 정의된 블록을 활용하여 로봇 하드웨어와 1대1로 대칭되는 Device 객체로 정의합니다.
3. 정의한 각 Device 객체 블록들을 서로 결합하여 하나의 모델을 만들어냅니다. 이와 같이 로봇 하드웨어를 추상화하여 소프트웨어로써 모델링해 얻은 모델을 **Roboid** 라고 합니다.
4. Roboid 내부의 각 블록들은, 사전 정의된 규칙에 따라 JSON 데이터 객체로 변환됩니다.
5. 하위 클래스 객체가 상위 클래스 객체의 특성을 상속받는 구조로 연결되어, 최종적으로 트리 구조를 갖는 하나의 **JSON 데이터**로 변환됩니다.
만약 A 블록 내부에 B 블록이 들어 있다면, A 블록은 B 블록의 상위(부모) 클래스 객체가 되고,
B 블록은 A 블록의 하위(자식) 클래스 객체가 되며, A 블록의 특성을 상속받습니다.

Roboid 클래스 정의

Roboid를 구성하는 Device 객체들의 클래스는 각각 다음과 같습니다.

1. Roboid



Roboid 클래스는 최상위 부모 클래스로, 어떤 로보이드(로봇)인지 나타냅니다.

Roboid 클래스는 하나 이상의 unit 또는 Device 를 구성요소로 갖습니다.

name(이름), implement(모델명), pid(제품 고유 id) 속성을 갖습니다.

구분자는 : 입니다.

2. Module



Module 클래스는 Roboid 클래스의 기능을 확장하기 위한 모듈을 정의할 때 사용됩니다.

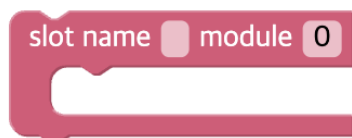
외부 통신은 roboid 클래스에 의존하지만, 기능적으로는 독립되어 있고 내부적으로 자체 인터페이스를 갖습니다. Roboid 클래스와 동일한 내부 구조를 가지며, 하나 이상의 unit 또는 Device를 구성요소로 갖습니다.

예시로, Raccoon 로봇 팔의 Peripheral(주변장치) 중 하나인 Conveyor는 내부적으로 자체 인터페이스를 갖고 있지만, 외부 통신은 Raccoon 에 의존하여 동작합니다.

name(이름), mid(모듈 고유 id) 속성을 갖습니다.

구분자는 + 입니다.

3. slot



Slot 클래스는 Module이 결합할 수 있는 인터페이스를 의미합니다.

사용 가능한 여러 Module 중 하나를 선택해서 사용할 때, Slot을 통해 구분할 수 있습니다.

name(이름), module(연결된 모듈 id) 속성을 갖습니다.

구분자는 > 입니다.

4. Unit



Unit 클래스는 특별한 기능 없이 단순히 이름을 구분하기 위해 사용되며, 비슷한 특성을 갖는 Device 들끼리 묶는 컨테이너로서의 역할을 합니다.

name(이름) 속성을 갖습니다.

구분자는 . 입니다.

- Notifier 는 데이터의 수신 성공 여부와 관계 없이 일정한 주기마다 단방향으로 정보를 전송하는 유형의 디바이스 클래스를 나타냅니다.

다음 조건 중 하나라도 해당하면 Notifier가 됩니다.

- 1) 시간에 따라 빠르게 계속 변화하는 장치
- 2) 중간 과정에 대한 수치적인 값이 의미있는 장치
- 3) 수신 성공 여부가 중요하지 않은 장치
- 4) 실행 과정 또는 완료 여부에 대한 정보가 필요하지 않은 장치

Notifier에 해당하는 Device 객체로는 Effector 와 Sensor 클래스가 있습니다.

5. Effector



```
effector type uint8 size 1 min 0 max 0 unit
```

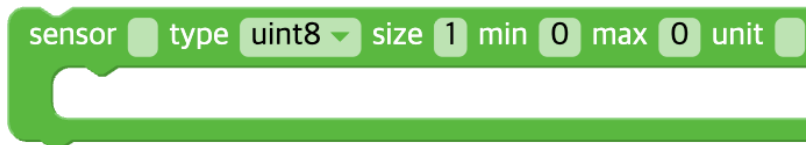
Effector 클래스는 로봇을 구성하는 실제 장치 중, 제어 가능한 장치를 나타냅니다.

예시로는 모터의 속도 값, LED의 밝기 값, 버저의 주파수 값 등이 있습니다.

name(이름), type(데이터 타입), size(데이터 크기), min(최솟값), max(최댓값), unit(단위) 속성을 갖습니다.

구분자는 없습니다.

6. Sensor



```
sensor type uint8 size 1 min 0 max 0 unit
```

Sensor 클래스는 로봇을 구성하는 실제 장치 중, 데이터를 생성하는 장치를 나타냅니다.

예시로는 온도, 가속도 센서, 엔코더 값 등이 있습니다.

name(이름), type(데이터 타입), size(데이터 크기), min(최솟값), max(최댓값), unit(단위) 속성을 갖습니다.

구분자는 없습니다.

- Responder 는 데이터의 수신 성공 여부 판단이 필요한 디바이스 클래스를 나타냅니다.

다음 조건 중 하나라도 해당하면 Responder가 됩니다.

- 1) 수신 성공 또는 완료 여부가 중요한 장치
- 2) 중간 값 또는 실행 과정에 대한 정보가 필요하지 않은 장치

Responder에 해당하는 Device 객체로는 Command 와 Event 클래스가 있습니다.

7. Command



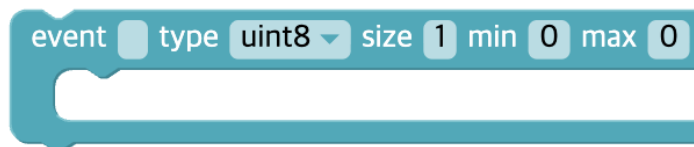
Command 클래스는 Effector와 같이 제어 가능한 장치를 나타내며, 로봇에 전달되어야 하는 제어 명령을 정의할 때 사용합니다.

Command 객체에 데이터를 씌으로써 로봇에 제어 명령을 전송할 수 있으며, 명령을 전송할 때마다 내부적으로 command_id 값이 증가하여, 명령 횟수를 확인할 수 있습니다.

name(이름), type(데이터 타입), size(데이터 크기), min(최솟값), max(최댓값) 속성을 갖습니다.

구분자는 없습니다.

8. Event



Event 클래스는 Sensor와 같이 데이터를 생성하는 장치를 나타내며, 반드시 확인해야 하는 특정 이벤트를 정의할 때 사용합니다.

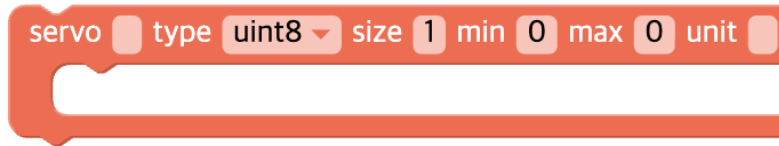
로봇으로부터 특정 이벤트에 대한 응답을 받을 때마다 내부적으로 event_id 값이 증가하고, 그 순간 e 값이 true로 전환되어 이벤트 발생 여부를 확인할 수 있습니다.

name(이름), type(데이터 타입), size(데이터 크기), min(최솟값), max(최댓값) 속성을 갖습니다.

구분자는 없습니다.

Servo, Action 클래스는 명령 수행이 완료되기까지 시간이 걸리는 장치를 나타냅니다.

9. Servo



Servo 클래스는 로봇이 특정 동작을 수행하도록 명령을 전달하고 실행 과정을 확인할 수 있으며, 해당 동작이 완료되었는지 여부를 확인해야 할 때 사용합니다. 예시로는 로봇 관절의 각도 설정, 일정 거리 또는 회전에 관한 명령 등이 있습니다.

name(이름), type(데이터 타입), size(데이터 크기), min(최솟값), max(최댓값), unit(단위) 속성을 갖습니다.

Servo 객체를 생성하면, 내부적으로 Event, Sensor 객체가 함께 생성됩니다.

생성된 Event 객체의 각 속성 값은 다음과 같습니다.

이름: !{name} / 데이터 타입: uint8 / 데이터 크기: 0 / 최솟값: 0 / 최댓값: 0

생성된 Sensor 객체는 중간 값을 출력하며, 실행 과정을 확인할 때 사용됩니다.

이름은 ~{name} 으로 설정되며, Servo 객체와 동일한 데이터 타입, 크기 속성을 갖습니다.

구분자는 없습니다.

10. Action



Action 클래스는 로봇이 특정 동작을 수행하도록 명령을 전달하고 해당 동작이 완료되었는지 여부를 확인해야 할 때 사용합니다.

name(이름), type(데이터 타입), size(데이터 크기), min(최솟값), max(최댓값) 속성을 갖습니다.

Action 객체를 생성하면, 내부적으로 Event 객체가 함께 생성됩니다.

생성된 Event 객체의 각 속성 값은 다음과 같습니다.

이름: !{name} / 데이터 타입: uint8 / 데이터 크기: 0 / 최솟값: 0 / 최댓값: 0

구분자는 없습니다.

11. constant

constant name value

Constant 클래스는 각 Device 객체가 가질 수 있는 기본적으로 가질 수 있는 상수 값을 미리 정의할 때 사용합니다.

name(이름), value(값) 속성을 갖습니다.

객체의 name 앞에 ^ 가 붙습니다

구분자는 없습니다.

데이터 타입(DataType)

type	bytes per size	description	c type
UInt8	1	8-bit unsigned integer	unsigned char
Int8	1	8-bit signed integer	signed char
UInt16	2	16-bit unsigned integer	unsigned short
Int16	2	16-bit signed integer	signed short
UInt32	4	32-bit unsigned integer	unsigned int
Int32	4	32-bit signed integer	signed int
Float32	4	32-bit floating point	float
Text	1	text	char[]

고유 식별자, urn

각 Roboid 와 Device 객체들은 고유한 식별자인 urn 을 갖습니다.

각 객체들의 urn은 다음과 같은 규칙을 기반으로 결정됩니다.

{ Roboid 클래스 객체의 urn }

= { implement } + '*' + { 인덱스 번호 } + { roboid 객체의 구분자 }

{ Device 객체의 urn }

= { 부모 Device 객체의 urn } + { Device 객체의 name } + { Device 객체의 구분자 }

로봇을 1대만 사용할 경우, 인덱스 번호는 0입니다.

멀티 로봇을 사용하는 경우, 같은 종류의 로봇의 개수만큼 인덱스 번호가 증가합니다.

(0, 1, 2, ...)

RobomationLAB 로봇 코딩 기본 문법 체계

RobomationLAB에서 제공하는 로봇 코딩 프로그램에서 코드를 작성할 때 지켜야 할 기본 문법 체계는 다음과 같습니다.

1. 로봇 Device 객체

로봇 Device 객체는 언어에 따라 다음과 같은 방법으로 호출할 수 있습니다.

- 자바스크립트: `${Device 객체 urn}`
- 파이썬: `__({Device 객체 urn})`

이 객체를 활용해, 값을 읽거나 쓰는 코드를 작성할 수 있습니다.

2. d 함수

d 함수는 Device 객체의 데이터(값)을 다루는 함수입니다.

값을 읽을 수도 있고, 값을 쓸 수도 있습니다.

Effector 클래스 객체의 값을 설정하거나, Command 클래스 객체에 명령을 입력할 때 주로 사용됩니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 1) 값 읽기
 - 자바스크립트: `${Device 객체 urn}.d`
 - 파이썬: `__({Device 객체 urn}).d`
- 2) 값 쓰기
 - 자바스크립트: `${Device urn}.d = {데이터};`
 - 파이썬: `__({Device 객체 urn}).d = {데이터}`

3. c 함수

c 함수는 Device 객체의 명령 호출 횟수를 다루는 함수입니다.

값을 읽는 것만 가능하며, Device 객체에 값이 쓰여진 횟수를 반환합니다.

Command 클래스 객체를 통해 특정 동작이 몇번 수행됐는지 확인할 때 주로 사용됩니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `${Device 객체 urn}.c`
- 파이썬: `__({Device 객체 urn}).c`

4. e 함수

e 함수는 Device 객체의 이벤트 처리를 다루는 함수입니다.

Event 클래스 객체에 한해서 값을 읽는 것만 가능하며, Event 객체에 대해 상태 변화나 환경의 변화가 생길 때 발생하는 이벤트를 감지하여, 이벤트가 발생한 순간에 true 값을 반환합니다.

모든 Device 객체들의 e 값은 loop 함수가 시작되기 전에 설정되고,

매 loop 함수의 수행이 완료되면 자동으로 false 로 리셋됩니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `${'Device 객체 urn'}.e`
- 파이썬: `__({'Device 객체 urn'}).e`

5. w 함수

w 함수는 Device 객체의 동작 완료 이벤트를 기다리는 함수입니다.

Action 또는 Servo 클래스 객체에서 파생된 Event 클래스 객체에 한해서, 특정 동작이 완료되었다는 이벤트를 확인할 때까지 기다립니다.

이벤트를 확인하고 난 뒤에 다음 코드를 이어서 수행합니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `await ${'Device 객체 urn'}.w();`
- 파이썬: `await __({'Device 객체 urn'}).w()`

● 주의사항

w 함수는 특정 동작이 완료될 때까지 기다리는 역할을 하기 때문에, await 키워드를 함께 사용합니다.

이때, await 키워드가 붙은 코드는 반드시 async 함수 내에서만 사용해야 합니다.

따라서, RobomationLAB의 코딩 프로그램 내에서 w 함수를 사용할 때는,

반드시 async 함수인 setup 내에서 사용해야 하며, loop 내에서는 사용해서는 안됩니다.

RobomationLAB 커스텀 함수

1. 기본 유틸리티 함수

● 기다리기

순차 실행 방식으로 코드를 실행할 때 사용하는 시간 지연 함수입니다.

매개 변수로 입력한 시간만큼 기다린 이후에 다음 코드를 수행합니다.

앞에 반드시 `await` 키워드가 붙어야 하며, `async` 가 붙은 함수 내에서만 사용할 수 있습니다.

`loop` 함수 내에서는 사용할 수 없습니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `await __wait(1000);` // (milli-seconds)
- 파이썬 : `await asyncio.sleep(1)` # (seconds)

● 키 다운

키보드 입력을 감지하여, 매개변수로 입력한 `keyCode`와 마지막에 누른 키의 일치 여부를 반환하는 함수입니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `__keydown(0)` // (keyCode)
- 파이썬: `__keydown(0)` # (keyCode)

● 키 업

키보드 입력을 감지하여, 매개변수로 입력한 `keyCode`와 마지막에 손을 떼 키의 일치 여부를 반환하는 함수입니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `__keyup(0)` // (keyCode)
- 파이썬: `__keyup(0)` # (keyCode)

● 키 입력

키보드 입력을 감지하여, 매개변수로 입력한 `keyCode` 배열과 현재 눌러 있는 키들이 일치하는지 여부를 반환하는 함수입니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `__keypressed([0, 0, 0, 0, 0])` // (keyCode list)
- 파이썬: `__keypressed([0, 0, 0, 0, 0])` # (keyCode list)

● 로그

콘솔 - 로그에 특정 변수의 값을 출력하는 함수입니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `__log(0, "", "");` // (signal, title, unit)
- 파이썬 : `__log(0, "", "")` # (signal, title, unit)

● 스코프

콘솔 - 스코프에 특정 변수 값의 변화를 그래프로 출력하는 함수입니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `__scope("", 0, 1, '#000000', 0);` // (title, min, max, color, signal)
- 파이썬 : `__scope("", 0, 1, '#000000', 0)` # (title, min, max, color, signal)

● 무작위 색상

무작위 RGB 값을 갖는 [red, green, blue] 배열을 반환하는 함수입니다.

언어에 관계 없이 다음과 같이 코드를 작성합니다.

`__randomColor()`

● 소리 재생

프로그램에 등록된 소리를 PC 또는 디바이스를 통해 재생하는 함수입니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `__playSound("", 100, false);` // (name, volume, repeat)
- 파이썬: `__playSound("", 100, True)` # (name, volume, repeat)

● 다음을 말하기

텍스트를 음성으로 변환하여 PC 또는 디바이스를 통해 재생하는 함수입니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `__speak("");` // (text)
- 파이썬: `__speak("")` # (text)

● 종료하기

코드 실행을 종료하는 함수입니다.

언어에 관계 없이 다음과 같이 코드를 작성합니다.

`__exit()`

2. 로봇 별 특수 기능 함수

● 바퀴 속도 (speed)

0 ~ 100 사이의 값으로 입력한 로봇의 바퀴 속도를 실제 로봇이 이동하기 위해 필요한 값으로 변환해주는 함수입니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `__getSpeed("", 50)` // (robot, value)
- 파이썬 : `__getSpeed("", 50)` # (robot, value)

이 함수를 사용하는 로봇:

햄스터 S(HamsterS), 햄스터(Hamster), 터틀(Turtle), 뽀오(Pio), 비글(Beagle)

● 이동 거리 (distance)

30cm, 500mm 와 같이 입력한 이동 거리를, 로봇이 실제로 이동하기 위해 필요한 값으로 변환해주는 함수입니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `__getDistance("", 50, 'cm')` // (robot, value, unit)
- 파이썬 : `__getDistance("", 50, 'cm')` # (robot, value, unit)

이 함수를 사용하는 로봇:

햄스터 S (HamsterS), 터틀 (Turtle), 뽀오 (Pio), 비글 (Beagle),

라쿤 주변장치 - 컨베이어, 슬라이더 (Conveyor, Slider)

● 왼쪽으로 제자리 돌기 (turn left)

제자리에서 왼쪽으로 n도 만큼 회전하는 함수입니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `__turn_degree_left("", 90, false);` // (robot, degree, wait_w)
- 파이썬 : `await __turn_degree_left("", 90, True)` # (robot, degree, wait_w)

이 함수를 사용하는 로봇:

햄스터 S(HamsterS), 터틀(Turtle), 뽀오 (Pio), 비글(Beagle)

wait_w 값이 true이면, await 키워드를 함께 사용하여 회전이 완료될 때까지 기다리다가 멈춘 뒤 다음 코드를 수행합니다. wait_w 값이 false 이면, 다음 코드를 이어서 수행합니다.

● 오른쪽으로 제자리 돌기 (turn right)

제자리에서 오른쪽으로 n도 만큼 회전하는 함수입니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `__turn_degree_right("", 90, false);` // (robot, degree, wait_w)
- 파이썬 : `await __turn_degree_left("", 90, True)` # (robot, degree, wait_w)

이 함수를 사용하는 로봇:

햄스터 S(HamsterS), 터틀(Turtle), 뽀오 (Pio), 비글(Beagle)

wait_w 값이 true이면, await 키워드를 함께 사용하여 회전이 완료될 때까지 기다리다가 멈춘 뒤 다음 코드를 수행합니다. wait_w 값이 false 이면, 다음 코드를 이어서 수행합니다.

- **터보 (turbo)**

로봇의 터보 모드를 활성화하거나 비활성화하는 함수입니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `__turbo("", false);` // (robot, turbo)
- 파이썬 : `__turbo("", True)` # (robot, turbo)

이 함수를 사용하는 로봇:

뽀오 (Pio)

- **정지하기 (stop move)**

바퀴 이동을 멈추는 함수입니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `__stopMove("");` // (robot)
- 파이썬 : `__stopMove("")` # (robot)

이 함수를 사용하는 로봇:

햄스터 S(HamsterS), 햄스터(Hamster), 터틀(Turtle), 뽀오(Pio), 비글(Beagle)

- **n초 후에 정지하기 (stop after delay)**

특정 시간만큼 기다린 이후에 바퀴 이동을 멈추는 함수입니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `__stopAfterDelay("", 5, false);` // (robot, delay, wait_w)
- 파이썬 : `await __stopAfterDelay("", 5, True)` # (robot, delay, wait_w)

이 함수를 사용하는 로봇:

햄스터 S(HamsterS), 햄스터(Hamster), 터틀(Turtle), 뽀오(Pio), 비글(Beagle)

wait_w 값이 true이면, await 키워드를 함께 사용하여 시간이 지날 때까지 기다리다가 바퀴 이동을 멈춘 뒤 다음 코드를 수행합니다. wait_w 값이 false 이면, 다음 코드를 수행하다가, 지정한 시간이 지나면 바퀴 이동을 멈춥니다.

- **말판 왼쪽으로 회전하기 (grid turn left)**

말판 위에서 왼쪽으로 1번 회전하는 함수입니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: **__grid_turn_left("", false);** // (robot, wait_w)
- 파이썬 : **await __grid_turn_left("", True)** # (robot, wait_w)

이 함수를 사용하는 로봇:

햄스터 S(HamsterS), 햄스터(Hamster), 삐오(Pio)

wait_w 값이 true이면, await 키워드를 함께 사용하여 회전이 완료될 때까지 기다리다가 멈춘 뒤 다음 코드를 수행합니다. wait_w 값이 false 이면, 다음 코드를 이어서 수행합니다.

● 말판 오른쪽으로 회전하기 (grid turn right)

말판 위에서 오른쪽으로 1번 회전하는 함수입니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: **__grid_turn_right("", false);** // (robot, wait_w)
- 파이썬 : **await __grid_turn_right("", True)** # (robot, wait_w)

이 함수를 사용하는 로봇:

햄스터 S(HamsterS), 햄스터(Hamster), 삐오(Pio)

wait_w 값이 true이면, await 키워드를 함께 사용하여 회전이 완료될 때까지 기다리다가 멈춘 뒤 다음 코드를 수행합니다. wait_w 값이 false 이면, 다음 코드를 이어서 수행합니다.

● 말판 앞으로 이동하기 (grid move forward)

말판 위에서 앞으로 1칸 이동하는 함수입니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: **__grid_move_forward("", false);** // (robot, wait_w)
- 파이썬 : **await __grid_move_forward("", True)** # (robot, wait_w)

이 함수를 사용하는 로봇:

햄스터 S(HamsterS), 햄스터(Hamster), 삐오(Pio)

wait_w 값이 true이면, await 키워드를 함께 사용하여 이동이 완료될 때까지 기다리다가 멈춘 뒤 다음 코드를 수행합니다. wait_w 값이 false 이면, 다음 코드를 이어서 수행합니다.

● 말판 뒤로 이동하기 (grid move backward)

말판 위에서 뒤로 1칸 이동하는 함수입니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: **__grid_move_backward("", false);** // (robot, wait_w)
- 파이썬 : **await __grid_move_backward("", True)** # (robot, wait_w)

이 함수를 사용하는 로봇:

삐오(Pio)

wait_w 값이 true이면, await 키워드를 함께 사용하여 이동이 완료될 때까지 기다리다가 멈춘 뒤 다음 코드를 수행합니다. wait_w 값이 false 이면, 다음 코드를 이어서 수행합니다.

- **말판 왼쪽으로 이동하기 (grid move left)**

말판 위에서 왼쪽으로 1칸 이동하는 함수입니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `__grid_move_left("", false);` // (robot, wait_w)
- 파이썬 : `await __grid_move_left("", True)` # (robot, wait_w)

이 함수를 사용하는 로봇:

삐오(Pio)

wait_w 값이 true이면, await 키워드를 함께 사용하여 이동이 완료될 때까지 기다리다가 멈춘 뒤 다음 코드를 수행합니다. wait_w 값이 false 이면, 다음 코드를 이어서 수행합니다.

- **말판 오른쪽으로 이동하기 (grid move right)**

말판 위에서 오른쪽으로 1칸 이동하는 함수입니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `__grid_move_right("", false);` // (robot, wait_w)
- 파이썬 : `await __grid_move_right("", True)` # (robot, wait_w)

이 함수를 사용하는 로봇:

삐오(Pio)

wait_w 값이 true이면, await 키워드를 함께 사용하여 이동이 완료될 때까지 기다리다가 멈춘 뒤 다음 코드를 수행합니다. wait_w 값이 false 이면, 다음 코드를 이어서 수행합니다.

- **축 기준으로 제자리 돌기 (pivot)**

제자리에서 특정 축을 기준으로 원하는 방향으로 n도 만큼 회전하는 함수입니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `__pivot("", "", "", 90, false);`
// (robot, unit, direction, degree, wait_w)
- 파이썬 : `await __pivot("", "", "", 90, True)`
(robot, unit, direction, degree, wait_w)

이 함수를 사용하는 로봇:

햄스터 S(HamsterS), 터틀(Turtle)

unit 은 로봇 별로 다음과 같은 값을 가질 수 있습니다.

- 햄스터 S: left pen, right pen, left wheel, right wheel
- 터틀: left wheel, right wheel

direction 은 로봇에 관계 없이 다음과 같은 값을 가질 수 있습니다.

forward, backward

wait_w 값이 true이면, await 키워드를 함께 사용하여 회전이 완료될 때까지

기다리다가 멈춘 뒤 다음 코드를 수행합니다. wait_w 값이 false 이면, 다음 코드를 이어서 수행합니다.

● 축 기준으로 원 그리며 돌기 (pivot circle)

특정 축을 기준으로 반지름이 k cm인 원을 그리며 원하는 방향으로 n도 만큼 회전하는 함수입니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `__pivot("", "", "", __getDistance("", 1, 'cm'), 90, false);`
// (robot, unit, direction, radius, degree, wait_w)
- 파이썬 : `await __pivot("", "", "", __getDistance("", 1, 'cm'), 90, True)`
(robot, unit, direction, radius, degree, wait_w)

이 함수를 사용하는 로봇:

햄스터 S(HamsterS), 터틀(Turtle)

unit 은 로봇 별로 다음과 같은 값을 가질 수 있습니다.

- 햄스터 S: left pen, right pen
- 터틀: "

direction 은 로봇에 관계 없이 다음과 같은 값을 가질 수 있습니다.

left forward, left backward, right forward, right backward

wait_w 값이 true이면, await 키워드를 함께 사용하여 회전이 완료될 때까지

기다리다가 멈춘 뒤 다음 코드를 수행합니다. wait_w 값이 false 이면, 다음 코드를 이어서 수행합니다.

● 소리 끄기 (stop sound)

소리 재생을 멈추는 함수입니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `__stopSound("");` // (robot)
- 파이썬 : `__stopSound("")` # (robot)

이 함수를 사용하는 로봇:

햄스터 S(HamsterS), 햄스터(Hamster), 터틀(Turtle), 삐오(Pio), 비글(Beagle),
라쿤(Raccoon)

- **순기구학 (angles to xyz)**

로봇 팔의 관절 각도를 통해, 말단 장치가 현재 위치한 좌표를 계산해 반환하는 함수입니다.

순기구학 (Forward Kinematics) 알고리즘을 사용합니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `_angles_to_xyz("", [0,0,0,0], "")` // (robot, angles, origin)
- 파이썬 : `_angles_to_xyz("", [0,0,0,0], "")` # (robot, angles, origin)

이 함수를 사용하는 로봇:

라쿤(Raccoon)

origin은 다음과 같은 값을 가질 수 있습니다.

wrist, end_effector

- **역기구학 (xyz to angles)**

말단 장치가 현재 위치한 좌표를 통해, 로봇 팔의 네 관절 각도를 계산해 반환하는 함수입니다.

역기구학 (Inverse Kinematics) 알고리즘을 사용합니다.

언어에 따라 다음과 같이 코드를 작성합니다.

- 자바스크립트: `_xyz_to_angles("", [0,100, 100], "")` // (robot, xyz, origin)
- 파이썬 : `_xyz_to_angles("", [0,100,100], "")` # (robot, xyz, origin)

이 함수를 사용하는 로봇:

라쿤(Raccoon)

origin은 다음과 같은 값을 가질 수 있습니다.

wrist, end_effector