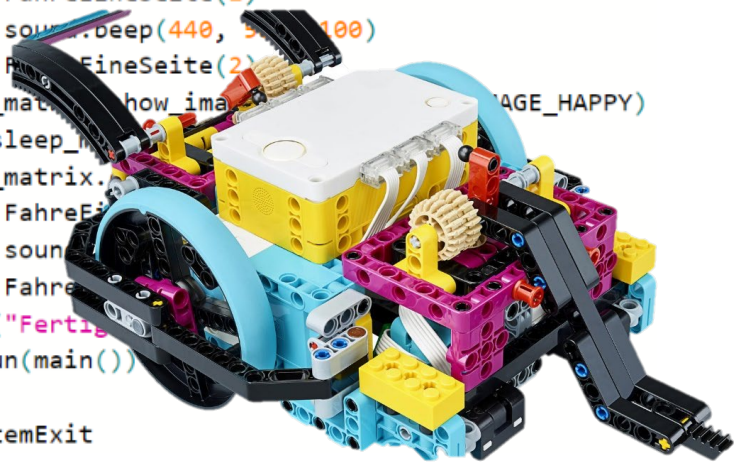
SWITZERLAND

PYTHON PER SPIKE PRIME

Nozioni di base

```
1  from hub import port, sound, light_matrix
2  import motor_pair, runloop, time
3
4  MP = motor_pair.PAIR_1
5  motor_pair.pair(MP, port.A, port.B)
6
7  async def FahreEineSeite(SEITENNUMMER):
8      light_matrix.write(str(SEITENNUMMER))
9      await motor_pair.move_for_degrees(MP, 360, 0)
10     await motor_pair.move_for_degrees(MP, 210, 100)
11     light_matrix.clear()
12
13 async def main():
14     await FahreEineSeite(1)
15     await sou.beep(440, 5   100)
16     await F  EineSeite(2)
17     light_mat  how_ima   AGE_HAPPY)
18     time.sleep_
19     light_matrix.
20     await FahreE
21     await soun
22     await Fahre
23     print("Fertig
24 runloop.run(main())
25
26 raise SystemExit
```



Vera Hausherr
Versione 1.1 / 13.10.2024

Copyright

Marchi registrati

LEGO®, SPIKE™ Prime, LEGO® Education, SPIKE™ App e altre denominazioni che siano relative a LEGO sono marchi registrati. Per rendere il testo più leggibile sono stati eliminati i simboli dei marchi.

Avvertenza tecnica

La presente guida è stata redatta sulla base della versione 3 dell'app SPIKE. Vi possono essere piccole differenze tra le figure presenti in questa guida e versioni precedenti (o successive non ancora pubblicate) dell'app SPIKE.

Voglia di più robotica?

Partecipa al concorso della World Robot Olympiad! Trovi tutte le informazioni su https://wro.swiss

Indice

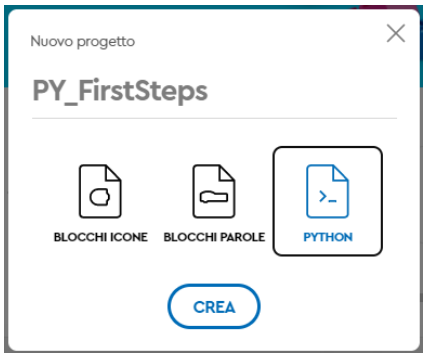
1. Principi

1.1 Creare un progetto

Obiettivi di apprendimento

- ☐ So come creare un nuovo progetto in Python.

- Apri l'app SPIKE e accedi alla sezione «SPIKE Prime».
- Nella pagina iniziale clicca su **Nuovo progetto** per creare un nuovo progetto.
- Suggerimento: accertati che sia selezionato **Python** e poi clicca su **Crea**.



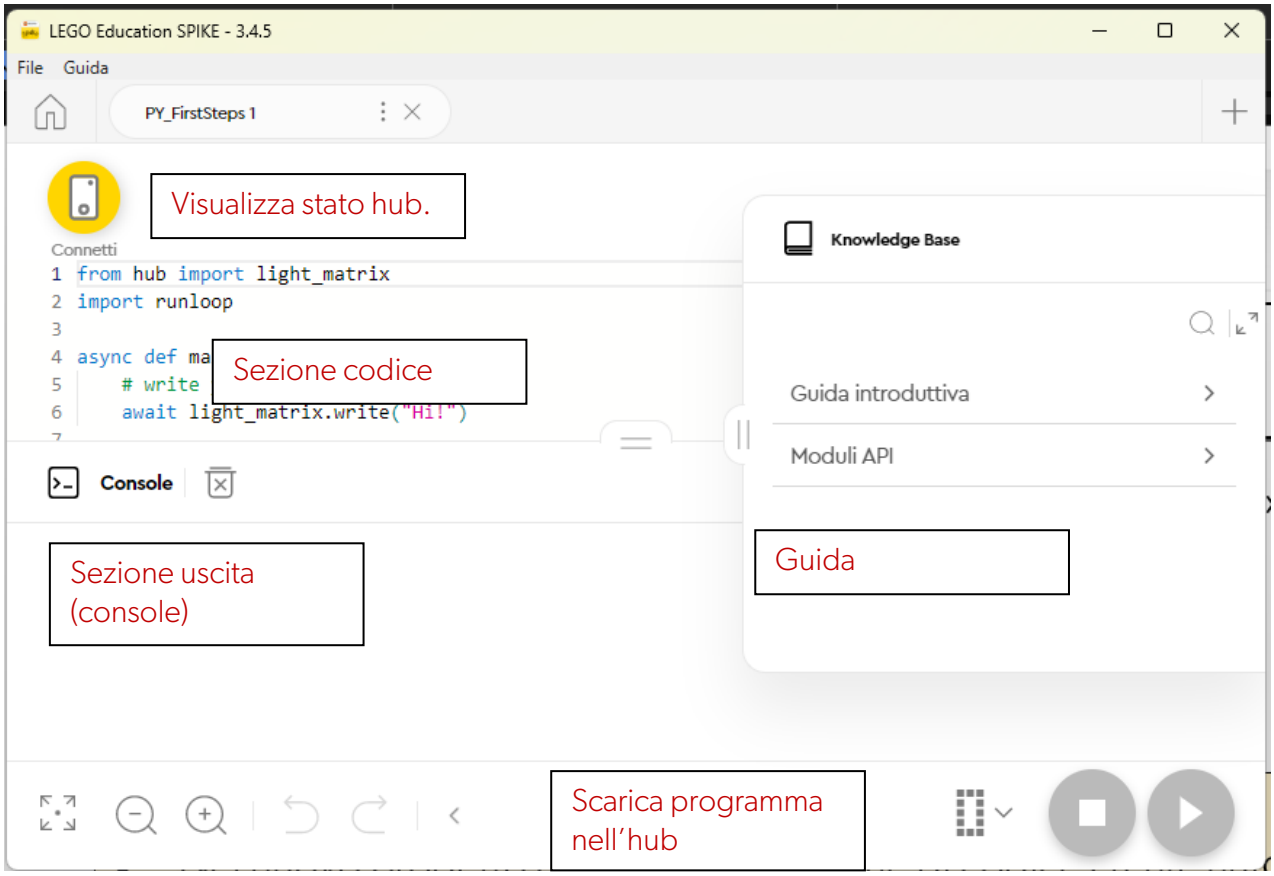
Suggerimento

Dai al progetto un nome che per te abbia un senso, per poterlo ritrovare facilmente in seguito.

Puoi anche stabilire una regola: inizia sempre i progetti Python con PY_, ad es. PY_FirstSteps. Quando successivamente li cercherai, troverai tutti i progetti Python nello stesso posto.

Se lavori a un grosso progetto puoi anche salvarlo con un nome sempre diverso, ad esempio aggiungendo la data, quindi PY_FirstSteps_2024-09-23. Così avrai un backup in caso di emergenza!

Ora vedi questa schermata:



Puoi ingrandire o rimpicciolire la sezione guida («Knowledge Base») e la sezione uscita («console») per avere più spazio per il codice.

Spiegheremo queste sezioni in seguito.

Compito 1.1

- Crea un progetto Python con il nome «PY_Hello_World».
- Nel nuovo progetto cancella tutte le righe di codice create automaticamente. Lavoreremo con il file vuoto.

1.2 Hello, World!

Obiettivo di apprendimento

☐ So com'è strutturato un programma semplice.

Adesso scriviamo il nostro primo programma. Quando si vuole provare un nuovo linguaggio di programmazione è buona norma creare un cosiddetto programma «Hello, World!», che fa proprio questo. All'avvio del programma sulla matrice luce dell'hub apparirà la frase «Hello, World!». Spesso è il programma più semplice che esista, in cui si può già vedere un po' come funziona il nuovo linguaggio.

Hai già cancellato il contenuto del file all'inizio. Ora ti resta solo una riga vuota già numerata. La numerazione delle righe viene creata automaticamente e non si può cancellare. Nei programmi più grandi noterai che è molto utile!

Adesso, nella prima riga scrivi il testo seguente: (Attenzione: maiuscole e minuscole devono essere esattamente come indicato qui sotto, altrimenti non funziona!):

```
from hub import light_matrix
```

In questo modo stai dicendo al programma che deve usare l'elemento «matrice luce». A volte ti servono anche altri elementi che dovrai importare.

Suggerimento

All'inizio, la questione dell'«import» può mandarti un po' in confusione. In effetti con Scratch (quasi) tutti gli elementi sono subito disponibili!

Ma Python è un sistema molto potente. Immagina una gigantesca biblioteca in cui si trovano libri su tantissimi argomenti diversi: geografia, storia, biologia, ecc. Se devi fare una presentazione sulle formiche, non hai bisogno di tutta la biblioteca ma solo dei libri sulle formiche, che trovi nella sezione «Biologia».

Nel nostro codice, **`hub`** è la «sezione» e **`light_matrix`** il «libro».

Grazie a Scratch conosci già un principio simile: cerchi in quale sezione colorata si trova il tuo blocco di testo e guardi il simbolo all'inizio del blocco. Per tutto quello che fai con la matrice luce ti servono i blocchi viola con il simbolo dell'hub:

Quindi, quando scrivi un programma, devi sempre pensare in quale sezione troverai i relativi elementi. La matrice luce è sull'hub; perciò importi da questa sezione.

Suggerimento

Nella Knowledge Base viene usata la parola «**modulo**» sia per la sezione che per l'elemento del programma.

In questa guida manteniamo i termini **sezione** ed **elemento** (di programma) per evitare confusione. La parola **modulo** viene utilizzata come termine generico per entrambi i concetti.

In questo momento il programma non sa ancora fare nulla. In fondo gli abbiamo solo detto quale elemento usare, ma non abbiamo ancora assegnato dei comandi a questo elemento.

Lo facciamo con un'altra riga di testo. Fai attenzione a scrivere esattamente così, senza spazi tra le parole e la punteggiatura, eccetto lo spazio tra «Hello,» e «World!».

```
3 light_matrix.write('Hello, World!')
```

Come vedi, viene scritto nella riga 3. La riga 2 resta vuota. Così separiamo la testa del codice (con i comandi **`import`**) **dal codice effettivo.**

Suggerimento

Il codice può presto diventare lungo e quindi poco chiaro. Abituati quindi a suddividerlo sempre con opportune righe vuote, per orientarti più facilmente!

Osserviamo ora la riga di codice 3.

Per prima cosa accediamo all'elemento **`light_matrix`** che abbiamo precedentemente importato. Poi diciamo cosa deve fare questo elemento: **`write()` (=scrivere).** Tra le parentesi inseriamo ora cosa deve essere scritto. Affinché il programma sappia che si tratta di un testo, aggiungiamo delle virgolette alte singole. Attenzione: con le virgolette doppie non funziona!

Nota bene

Abbiamo una struttura di base che riutilizziamo sempre in Python:

`elemento.comando(parametro)`
Un parametro è un valore variabile. Anziché `'Hello, World!'` potresti scrivere anche `'Ciao Mondo'`, `'Hallo Welt'` o qualsiasi altro testo. Non cambia nulla del programma in sé ma solo del risultato.

Se hai già lavorato con Scratch conosci bene i parametri: vengono sempre scritti nei campi ovali dei blocchi di testo.

Compito 1.2

- Cambia il codice in modo che ti saluti!

1.3 Commenti in Python

Obiettivo di apprendimento

☐ Sono in grado di commentare il mio codice e so perché è utile.

Se il tuo programma sa fare solo una cosa e questa cosa si capisce già dal nome del progetto, non occorre necessariamente che commenti il tuo codice. Spesso, però, i programmi diventano più lunghi e possono fare cose diverse.

In questo caso è opportuno continuare a scrivere cosa significa esattamente quello che hai appena codificato e perché l'hai programmato così.

I commenti iniziano sempre con il simbolo **#** e terminano alla fine della riga. Se vuoi scrivere un commento più lungo, ogni riga deve iniziare con **#**.

```
light_matrix.write('Hello, World!')
# L'hub scrive «Hello, World!»
# Se il commento è molto lungo,
# suddividilo in più righe
# e inizia ogni riga con #.
```

1.4 Colori nel codice

Obiettivo di apprendimento

☐ So quali sono i principali colori nel codice Python e come farne uso.

Avrai certamente già notato che il testo del tuo codice viene formattato automaticamente in colori diversi. Questo dimostra che il nostro ambiente di programmazione è alquanto furbo e capace di pensare.

Ecco i colori principali.

Blu Parole chiave che strutturano il nostro programma. Conosci già **from** e **import**, ma anche **if**, **else** (se, altrimenti) sono automaticamente illustrate in blu, come anche **True** e **False** (Vero e Falso; ricorda che queste due parole devono essere sempre scritte con l'iniziale maiuscola!).

Rosa Si tratta di parametri di testo che programmi come testo fisso, come nell'esempio **'Hello, World!'** di prima. Ci vogliono sempre le virgolette singole.

Arancione Sono parametri numerici che ti serviranno dopo, ad esempio se vuoi indicare distanze o velocità.

Verde Sono commenti che non influiscono sul programma, ma sono un aiuto per te, per capirci qualcosa anche dopo.

Nero Sono i nomi dei moduli (sezioni ed elementi) e dei comandi, nonché di variabili e funzioni. Anche la punteggiatura di solito è in nero, eccetto le parentesi **()**, che sono verde scuro.

Suggerimento

Aiuto, i miei colori sono sbagliati!

All'improvviso il tuo codice appare così?

```
3 light_matrix.write(Hello, World!')
```

Vuol dire che c'è un errore! Come sai, **Hello, World!** è un parametro di testo e deve essere rosa. Ma se guardi attentamente, vedrai che hai dimenticato una virgoletta! Basterà inserirla e il codice sarà di nuovo giusto.

Anche le linee ondulate rosse mostrano che nel tuo codice c'è un errore. Succede anche quando ad esempio sbagli a scrivere il nome di un comando.

```
3 light_martix.write('Hello, World')
```

Hai trovato l'errore?

1.5 Utilizzare un timer

Obiettivo di apprendimento

- ☐ So come utilizzare un timer per terminare un'azione dopo un tempo predeterminato.

Talvolta un'azione serve per una durata determinata. Ad esempio, sulla matrice luce deve apparire per due secondi esatti un cuore, che poi deve disattivarsi.

Ecco come si presenta il codice:

```
1 from hub import light_matrix
2 import time
3 # Ci serve un modulo che possa registrare il tempo. Si chiama time.
4 # time è un modulo a sé che non necessita di una sezione sovraordinata.
5
6 light_matrix.show_image(1)
```

```
7 # show_image() è il comando per visualizzare un'immagine della banca dati interna.
8 # L'immagine numero 1 è un cuore.
9
10 time.sleep_ms(2000)
11 # Questo è il comando di attesa.
12 # Richiamiamo il modulo time a cui appartiene il comando sleep_ms().
13 # Il parametro è il tempo di attesa in millisecondi.
14 # Per attendere 2 secondi dobbiamo quindi immettere 2000.
15
16 light_matrix.clear()
17 # Con questo comando si disattiva di nuovo la matrice luce.
18 # Qui non ci occorre alcun parametro, perché non ci sono diverse
19 # varianti di «spento». Ma le parentesi non si possono eliminare!
```

Compito 1.5

- Verifica quali immagini vengono visualizzate se cambi il parametro nella riga 6!
- Cambia il parametro nella riga 10. Qual è il numero più piccolo che puoi immettere per vedere ancora l'immagine?

2. Motori

Qui si fa interessante: usiamo i motori! In fin dei conti il nostro robot deve saper lavorare e muoversi!

Cominciamo con un singolo motore che ci occorre, ad esempio, per un utensile con cui si può raccogliere un oggetto.

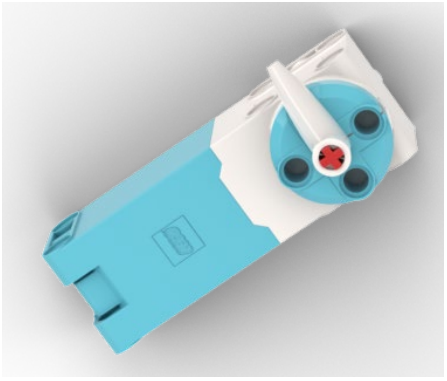
2.1 Un motore singolo

Obiettivo di apprendimento

- ☐ Sono in grado di usare un motore singolo e conosco i comandi e i parametri principali.

Vogliamo usare un motore singolo.

Connetti un motore medio alla **porta A** e applica sulla rotellina azzurra del motore un altro mattoncino Lego, in modo da poter osservare più facilmente le rotazioni.



Adesso scriviamo il codice per far girare questo motore. Precisamente deve fare tre rotazioni complete alla velocità di due rotazioni al secondo (U/s).

Per prima cosa dobbiamo importare i moduli necessari:

```
import motor
from hub import port
```

L'elemento **motore** contiene tutti i comandi per un motore singolo. L'elemento **port** controlla gli attacchi (porte).

E adesso tocca al comando; ha la forma seguente:

```
motor.run_for_degrees(porta, gradi, velocità)
```

Suggerimento

Questo comando ha tre parametri. Quando nella presente guida vengono utilizzate parole in italiano negli esempi di codice, si tratta di definizioni di aiuto per i parametri da selezionare. Ci aiutano ad annotarci cosa ci occorre. Per rendere ancora più evidente che non si tratta di veri e propri codici, nella guida queste parole sono scritte anche in ***corsivo e viola.***

- La ***porta*** la indichiamo con **port.A** o **port.B**, a seconda di quale delle due ci occorre.
- ***Gradi***: questo parametro mostra quanto deve ruotare il motore. Magari dobbiamo fare qualche calcolo: una rotazione corrisponde a 360 gradi (=360°) Quindi, se il motore deve fare tre rotazioni complete, il nostro secondo parametro sarà `1080` (3 × 360 = 1080).

- La ***velocità*** indica la rapidità di rotazione del motore. La velocità è indicata in gradi al secondo (°/s). Poiché sappiamo che una rotazione corrisponde a 360°, possiamo anche calcolare: 1 U/s = 360°/s; e rilevante per noi: 2 U/s = `720`°/s.
- **Tutti i parametri vengono immessi senza simboli!**

Il nostro codice adesso si presenta così:

```
import motor
from hub import port

motor.run_for_degrees(port.A, 1080, 720)
# Osserva i tre parametri!
```

Compito 2.1

- Sperimenta con parametri diversi. Per i parametri ***gradi*** e ***velocità*** prova a usare anche numeri negativi, alternamente e contemporaneamente, ad esempio `-360`**.** Cosa accade ora?

2.2 Due motori independenti

Obiettivo di apprendimento

☐ Sono in grado di gestire due motori indipendenti e so come farli funzionare contemporaneamente o uno dopo l'altro.

Adesso proviamo qualcosa di diverso: colleghiamo due motori contemporaneamente! Il primo motore resta collegato alla **porta A** mentre il secondo lo colleghiamo alla **porta B**. Entrambi i motori devono funzionare autonomamente perché ad esempio vengono utilizzati per due diversi utensili. Non si tratta quindi dei due motori di azionamento di un robot mobile.

Il nuovo codice si presenta così:

```
import motor
from hub import port

motor.run_for_degrees(port.A, 360, 720)
motor.run_for_degrees(port.B, 360, 720)
```

Cosa succede?

In effetti si potrebbe supporre che il motore connesso alla porta A faccia per primo una rotazione, seguito dal motore della porta B. Invece i due motori si muovono contemporaneamente.

Nota bene

`Run_for_degrees()` è un comando che non deve necessariamente essere completato prima che venga avviato il comando successivo. Al contrario, quest'ultimo viene avviato subito dopo l'avvio del primo. Dato che l'intervallo di tempo è minimo, all'occhio umano sembra che l'avvio sia contemporaneo, mentre in realtà c'è un intervallo di un paio di millisecondi.

I comandi che funzionano in questo modo, cioè che non devono prima essere completati, si chiamano **comandi** «**awaitable**». Significa che di norma i due

comandi funzionano in contemporanea, ma puoi anche aspettare che uno sia eseguito completamente prima di passare al successivo.

Difficile da capire? Prova a immaginare la vita quotidiana: vorresti ascoltare un brano e fare anche un videogioco sul cellulare. Puoi fare entrambe le cose contemporaneamente: mentre giochi puoi anche ascoltare la musica. Ma puoi anche prima ascoltare il brano e poi giocare.

Come si fa per far eseguire i due comandi in sequenza, cioè prima il motore sulla **porta A** e poi quello sulla **porta B**?

Si può. Ci serve innanzitutto un ciclo per il nostro programma:

```
1 import runloop
2
3 async def main():
4     # Scrivi qui il tuo programma
5
6 runloop.run(main())
```

Riga 1: per prima cosa, oltre agli altri moduli necessari, importiamo l'elemento **`runloop`.** Questo contiene le parole chiave che ci consentono di stabilire se i comandi «awaitable» devono essere eseguiti contemporaneamente o in sequenza.

Riga 3: il prossimo passo è definire una cosiddetta funzione. Un programma complesso è spesso suddiviso in più parti, definite funzioni. Cosa sono esattamente le funzioni lo scopriremo più avanti al capitolo 5. Per il momento ti serve solo la funzione **`main().`**

La funzione che ci serve qui si chiama **`main()`** e utilizza la parola chiave **`async`**, un'abbreviazione del termine inglese *asynchronous*, che significa «asincrono» (non contemporaneamente). Questa parola chiave ci segnala che la funzione contiene comandi «awaitable», che vengono eseguiti in sequenza. La parola chiave **`def`** segnala che il codice seguente costituisce la definizione della funzione, vale a dire che qui, dopo questa riga, ci sono tutti i comandi di questa funzione.

Il contenuto della funzione lo scriviamo `indentato` nel punto in cui al momento c'è il commento
`    # Scrivi qui il tuo programma`
in verde.

Riga 6: a questo punto richiamiamo la funzione, ovvero diciamo al programma che adesso deve essere eseguita. Questo richiamo non è più indentato.

Il codice completo si presenta ora così:

```
1  import motor
2  import runloop
3  from hub import port
4
5  async def main():
6      await motor.run_for_degrees(port.A, 360, 720)
7      # Con la parola chiave await il codice viene bloccato
8      # finché questa riga è stata eseguita fino in fondo.
```

```
9       await motor.run_for_degrees(port.B, 360, 720)
10 runloop.run(main())
```

Riga 6: qui vedi la parola chiave **`await`.** Questa freeza il codice finché il comando di questa riga non è completamente eseguito. Solo quando il motore **A** ha completato la rotazione, inizia la rotazione del motore **B**.

Compito 2.2

- Completa ora il codice, in modo che i due motori funzionino prima in sequenza e poi di nuovo contemporaneamente (soluzione alla fine della guida).

2.3 Motori di azionamento di un robot mobile («coppia di motori»)

Obiettivo di apprendimento

☐ So cosa sono i motori di azionamento e sono in grado di impostarli correttamente.

Se abbiamo un robot mobile ci occorrono due motori che lavorino insieme. Per farlo dobbiamo importare un'altra sezione, la **`motor_pair`** (coppia di motori). Ora questi due motori sono considerati un'unità e possono essere gestiti con un comando comune a entrambi.

Suggerimento
Se finora hai lavorato con Scratch, conosci già la differenza: un motore singolo viene gestito con blocchi blu, mentre una coppia di motori con blocchi rosa.

Adesso dobbiamo impostare i nostri motori di azionamento. In Scratch si presentava così:

In Python si presenta così:

```
1 from hub import port
2 import runloop
3 import motor_pair
4
5 async def main():
6     # Accoppia i motori sulle porte A e B:
7     motor_pair.pair(motor_pair.PAIR_1, Porta_Motore_Sinistra, Porta_Motore_Destra)
8     # Codice per ciò che adesso deve fare la coppia di motori
9
10 Runloop.run(main())
```

La riga importante è la **riga 7**. Indica quali sono le due porte che ci occorrono. Osserviamo questa riga pezzo per pezzo:

`motor_pair.pair()`: con questo comando i due motori vengono accoppiati e da questo momento in poi sono un'unità. Il comando ha tre parametri.

- **`motor_pair.PAIR_1`**: questo parametro resta sempre uguale. L'unica eccezione è quando il tuo robot ha più di due ruote con cui può muoversi e occorre passare da un'opzione all'altra. In tal caso su **`PAIR_2`** e **`PAIR_3`** potresti impostare anche altre coppie (questo caso però nella realtà si presenta piuttosto raramente; nonostante ciò, SPIKE Python esige che ogni volta che utilizzi la coppia di motori tu scriva che è la coppia **`PAIR_1`).**
- ***`Porta_Motore_Sinistra`***: indica qui la porta nella quale è inserito il motore di sinistra secondo il senso di marcia. Come abbiamo già visto prima, le porte vengono sempre indicate nel formato **`port.A`.**
- ***`Porta_Motore_Destra`***: indica qui la porta nella quale è inserito il motore di destra secondo il senso di marcia.

Compito 2.3

- Costruisci un robot semplice che, oltre all'hub, possiede due motori nelle porte A e B, una ruota collegata a ciascun motore e all'altra estremità la rotella a sfera turchese come sostegno. Se non riesci a farlo, la direzione del corso ha un modello.
- Crea un nuovo progetto con il nome **PY_Scuola_guida**.
- Importa i moduli necessari.
- Definisci una funzione **`async`** denominata **`main()`**.
- All'interno della funzione imposta la coppia di motori.
- Nel prossimo capitolo continueremo a lavorare con questo codice.

2.4 Guidare

Obiettivi di apprendimento

- ☐ So come far muovere il mio robot avanti e indietro.
- ☐ So sterzare.
- ☐ Conosco diversi metodi di arresto.

2.4.1 Guidare diritto

Ci sono diverse possibilità di guidare il robot.

La più semplice è guidare diritto per un certo tempo, ad esempio 2 secondi. Se non indico altro, il robot si muove alla velocità standard dei motori. È piuttosto lenta!

```
1 from hub import port
2 import runloop
3 import motor_pair
4
5 async def main():
6     # Accoppia due motori alle porte A e B
7     motor_pair.pair(motor_pair.PAIR_1, port.A, port.B)
8     # Aspetta un po' prima di avviare il motore; è utile per osservare
attentamente.
9     await runloop.sleep_ms(1000)
10
11    # Avvia il movimento alla velocità standard (piuttosto lenta)
12    motor_pair.move(motor_pair.PAIR_1, 0)
```

```
13
14    # Arresta il robot dopo 2 secondi
15    await runloop.sleep_ms(2000)
16    motor_pair.stop(motor_pair.PAIR_1)
17
18 runloop.run(main())
19
20 raise SystemExit
```

Ora esaminiamo di nuovo con più attenzione le singole righe. Eccezionalmente cominciamo dalla fine.

Riga 20: se ti sei già innervosito perché il programma non termina finché tu non premi il tasto, con questa riga puoi disattivare il programma. È opportuno avere questa come **ultima** riga di ogni programma, affinché il robot, dopo aver eseguito i comandi, passi automaticamente alla modalità standby e consumi meno batteria. D'ora in poi puoi tenere queste righe alla fine di ogni programma, anche se non viene specificato!

Riga 12: il comando **`motor_pair.move(`*`coppia di motori`*`, `*`sterzata`*`)`** ha due parametri obbligatori.

- ***`Coppia di motori`*** è la coppia di motori già definita in precedenza.
- *`Sterzata`* è scritto in arancione perché questo valore è sempre una cifra compresa tra `-100` e `100`. Questa cifra indica se e come ruota il robot. Se il parametro presenta il valore 0, il robot procede diritto. Potrai apprendere maggiori dettagli sui parametri di *`sterzata`* nel prossimo paragrafo.

Suggerimento

Se provi qualcosa di nuovo per la prima volta, usa delle cifre basse: percorsi e tempi brevi a basse velocità. Oppure fai muovere il tuo robot sul pavimento! Eviterai così che cada dal tavolo.

Ti sembra un po' troppo lento? Allora cambiamo la velocità!

Per farlo dobbiamo modificare la **riga 12**. Dobbiamo inserire nel comando **`move()`** un altro parametro per la velocità. Da

```
motor_pair.move(motor_pair.PAIR_1, 0)
```

diventa

```
motor_pair.move(motor_pair.PAIR_1, 0, velocity=velocità)
```

Come per il motore singolo, la velocità è indicata in gradi al secondo (°/s). La velocità massima del motore medio è di 1100°/s, cioè poco più di tre intere rotazioni al secondo. Se usi le ruote standard turchesi, corrisponde a 53,5 cm/s. Alla massima velocità il tuo robot percorre più di un metro in due secondi!

Compito 2.4.1

- Prova il codice con diverse velocità. Attenzione: più di 1100 non è tecnicamente possibile e risulta in un errore!

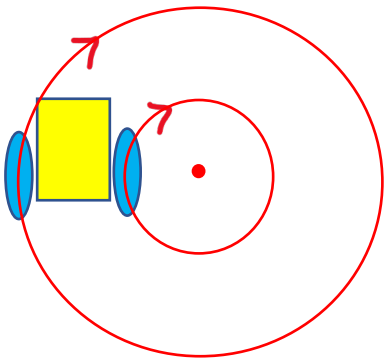
- Osserva se il robot avanza esattamente diritto. Va sempre diritto o fa un arco o una linea ondulata? C'è una velocità ideale alla quale viaggia meglio diritto?
- Osserva anche come si comporta il robot in frenata alle diverse velocità.
- Ora per una volta immetti anche delle cifre negative (tra 0 e -1100). Cosa succede?

2.4.2 Curve e rotazioni

Adesso non vogliamo più avanzare sempre solo diritto; vogliamo fare delle curve e ruotare. Per farlo dobbiamo modificare nel codice di prima alla riga 12 il parametro *`sterzata`, che finora era a `0`*. Indica quanto è forte la rotazione.

Per questo ci occorre un pochino di teoria. Infatti esistono tre tipi di curve o rotazioni.

Curva: puoi fare una curva (che di fatto è una parte di un cerchio):



In un simile caso per il parametro *`sterzata`* ci occorre una cifra compresa tra 0 e 50. Nell'esempio usiamo `30`.

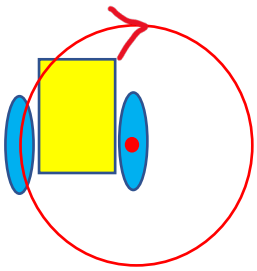
```
motor_pair.move(motor_pair.PAIR_1, 30, velocity=360)
```

Ecco cosa accade esattamente: un motore si muove un po' più velocemente dell'altro. Per effetto di questo movimento non uniforme si crea la rotazione.

Compito 2.4.2

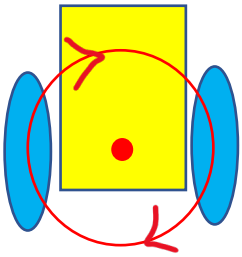
- Sperimenta cosa succede se aumenti o diminuisci il parametro *`sterzata`*!
- Ora immetti anche delle cifre negative tra 0 e -50. Cosa succede?

Rotazione circolare: una forma speciale di curva è la «rotazione circolare». Funziona esattamente come un cerchio in geometria: una ruota resta sul posto e l'altra gira.



Per la rotazione circolare il parametro *`sterzata`* deve essere esattamente 50 o -50 (a seconda della direzione in cui vogliamo ruotare).

Rotazione giroscopica: nella rotazione giroscopica il robot ruota al centro tra le due ruote motrici.



Questo è il tipo di rotazione che preferiamo. Il robot necessita di pochissimo spazio e possiamo impostare con la massima precisione il modo in cui deve stare dopo la rotazione. In questo caso il parametro *sterzata* è `100` o `-100`.

Tecnicamente, con questo tipo di rotazione, entrambe le ruote girano esattamente alla stessa velocità, ma nel senso opposto.

Per tutti i tipi di rotazione, come per l'avanzamento diritto, puoi fermare la rotazione dopo un tempo prestabilito e bloccare la coppia di motori.

2.5 Compiere un percorso determinato

Finora il tuo robot ha marciato per un tempo predefinito. Va bene, ma non è una soluzione molto pratica. Soprattutto se sappiamo esattamente quanto è lungo il percorso che deve fare, ci farebbe piacere avere un'altra possibilità.

Anziché il semplice comando **`move()`** per questo ci occorre il comando **`move_for_degrees()`**.

Il comando si presenta così:

```
motor_pair.move_for_degrees(motor_pair.PAIR_1, gradi, sterzata, velocity=velocità)
```

È importante soprattutto il secondo parametro, *gradi:* indica di quanti gradi deve ruotare il motore. Anche in questo caso si applica
1 rotazione = 360 gradi.

Esempio: il robot deve avanzare diritto esattamente per 45 cm e poi fermarsi. Come otteniamo i gradi partendo da 45 cm?

Qui ci serve un metro a nastro e dobbiamo fare un piccolo calcolo.

Con il metro misuriamo la circonferenza delle ruote collegate ai motori. Se usi le ruote standard del box Spike, non occorre misurare: la circonferenza è 17,5 cm. Per le altre ruote, però, devi assolutamente prendere la misura, perché a seconda delle dimensioni delle ruote cambia tutto!

Adesso devi calcolare: quante volte la circonferenza della ruota di 17,5 cm ci sta nel percorso di 45 cm?

$$\frac{45\ cm}{17{,}5\ cm} = 2.571$$

E come seconda operazione:

$$2.571 * 360° = 925.56°$$

Poiché per i gradi Python accetta solo numeri interi, dobbiamo arrotondare a 926° (e quando scriviamo il codice, dobbiamo tralasciare i gradi). Il codice adesso si presenta così:

```
motor_pair.move_for_degrees(motor_pair.PAIR_1, 926, 0, velocity=360)
```

Suggerimento

Per fare una curva o una rotazione funziona allo stesso modo. Il parametro *gradi* ti segnala quanto deve ruotare il motore più veloce. Sebbene si possa calcolare con la massima precisione, spesso nella quotidianità si fa prima a provare. Con il tempo si sviluppa una certa esperienza grazie alla quale impariamo la corrispondenza tra il valore *gradi* e la rotazione.

Un altro suggerimento: crea una tabella Excel, in cui inserisci i valori ottenuti con l'esperienza. Così avrai sempre questi parametri a portata di mano e potrai evitare di procedere per tentativi la prossima volta!

3. Sensori

3.1 Il sensore di colore

Obiettivo di apprendimento

- ☐ So come leggere e visualizzare i valori di un sensore di colore.

3.1.1 Colori

Il sensore di colore è in grado di misurare e riconoscere diversi tipi di valori: colori e luce riflessa. Osserviamo innanzitutto i colori.

Suggerimento

Il sensore è in grado di riconoscere solo i colori seguenti: rosso, verde, blu, magenta, giallo, arancione, turchese, azzurro, nero, bianco. Se ha davanti altri colori o più colori insieme, non li sa distinguere nettamente. Perciò la posizione del sensore rispetto all'oggetto e all'illuminazione dell'ambiente è molto importante. Il riconoscimento dei colori è una delle parti del codice che bisogna testare con maggiore frequenza, per evitare errori.

Inoltre, alcuni colori sono difficili da riconoscere per il sensore, in particolare l'arancione, il turchese e il rosa. Se possibile, cerca quindi di evitare questi colori.

Finché l'hub è collegato al computer/tablet, nell'indicatore di stato possiamo vedere quale colore sta riconoscendo il sensore.

In questo screenshot riconosce il blu sulla **porta C**. Come vedi, il blu ha il valore numerico 3, che ti servirà in seguito se, ad esempio, vuoi eseguire una determinata azione in una condizione quando il sensore riconosce il blu.

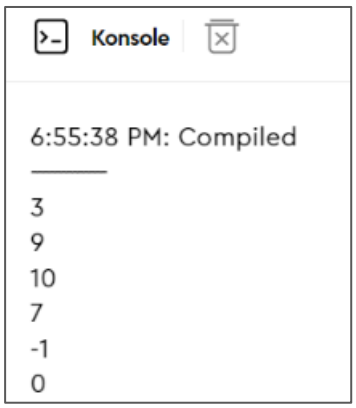
Esempio: adesso scriviamo un programma molto semplice con cui il robot riconosce sempre un colore, ne scrive il valore numerico nella sezione uscita (la console), attende due secondi (per darti il tempo di tenere un altro oggetto davanti al sensore) e poi riprende dall'inizio. Il programma viene eseguito finché non arresti l'hub con il pulsante grande.

```
1 from hub import port
2 import color_sensor
3 import runloop
4 import time
5
6 async def main():
7     while True:
8         # Scansiona il colore dell'oggetto e scrivi il valore nella console.
9         print(color_sensor.color(port.C))
10        # Aspetta due secondi e poi ripeti.
11        time.sleep_ms(2000)
12
13 runloop.run(main())
```

Riga 2: qui dobbiamo importare l'elemento **`color_sensor`.**

Riga 7: `while True` avvia un ciclo continuo. Impareremo a conoscere bene i cicli nel prossimo capitolo.

Riga 9: qui il valore del sensore viene riconosciuto e scritto nella console. Durante l'esecuzione il risultato si presenta ad esempio così:



Il risultato è da intendersi così: ogni riga è un passaggio di misurazione. Il numero è il valore numerico del colore riconosciuto; -1 significa che non è stato riconosciuto alcun colore.

Sono possibili questi valori:

- 0=nero
- 1=rosa
- 2=viola
- 3=blu
- 4=azzurro o turchese
- 6=verde
- 7=giallo
- 8=arancione
- 9=rosso
- 10=bianco

Nello screenshot precedente il sensore ha quindi riconosciuto in questo ordine i colori blu, rosso, bianco, giallo, nessun colore, nero.

Attenzione: alcuni colori vengono riconosciuti con difficoltà e generano un errore, anche se il programma è giusto. I colori più sicuri, oltre al nero e al bianco, sono il rosso, il giallo e il verde.

3.1.2 Luce riflessa

Ora utilizziamo il sensore di colore per richiamare la luce riflessa. Lo scriviamo di nuovo nella console.

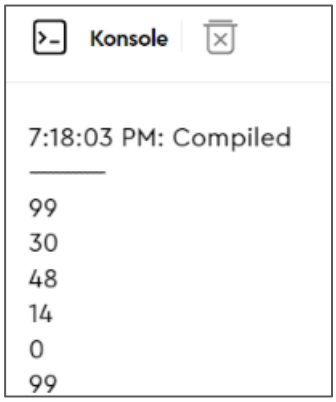
Quando il sensore misura la luce riflessa, restituisce valori da `0` a `100`. Possiamo interpretarli così: maggiore è il numero, più luce viene riflessa e più chiaro è l'oggetto; minore è il numero, più è scuro l'oggetto. Se nel campo di misura del sensore ci sono sia il bianco che il nero, il sensore misura il valore medio. Se tieni il sensore subito sopra il bordo tra qualcosa di bianco e qualcosa di nero, ottieni un valore pari a circa `50`. Questi valori ci torneranno utili, ad esempio, per un inseguitore di linea.

Ora dobbiamo modificare una riga dal codice per il riconoscimento dei colori del capitolo precedente.

```
9       print(color_sensor.reflection(port.C))
```

Durante il test tieni il sensore davanti a qualcosa di nero, qualcosa di bianco, qualcosa che contenga sia del nero che del bianco e sopra diverse gradazioni di grigio.

Il risultato potrebbe apparire così:



3.2 Il sensore di distanza

Obiettivo di apprendimento

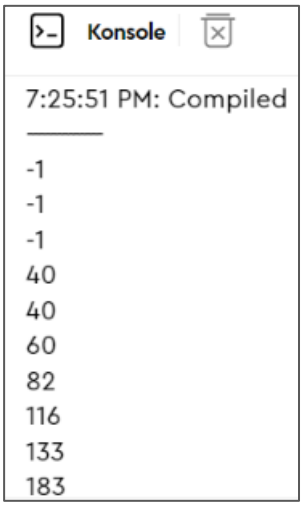
- ☐ Con il sensore di distanza sono in grado di leggere quanto dista il sensore da un ostacolo.

Il principio di questo compito è lo stesso del sensore di colore.

Esempio: leggiamo la distanza e la scriviamo nella console. Questa volta scegliamo però un intervallo di tempo più breve (200 ms = 0,2 s) poiché possiamo semplicemente tenere in mano il sensore e muoverlo.

```
1  from hub import port
2  import distance_sensor, runloop, time
3
4  async def main():
5      while True:
6          # Misura la distanza dal sensore di colore e scrivi il valore nella
console.
7          print(distance_sensor.distance(port.D))
8          # Aspetta 0,2 secondi e poi ripeti.
9          time.sleep_ms(200)
10
11 runloop.run(main())
```

Riga 7: qui si scrive nella console la distanza che il sensore misura sulla porta D.



Se qui si visualizza un valore di -1, significa che il sensore è troppo lontano dal suo obiettivo oppure troppo vicino. Le distanze consentite sono comprese tra 4 cm e 2 m. Un risultato di -1 si può avere anche quando il sensore si trova in un angolo così strano rispetto all'oggetto, che il sensore non sa dove misurare.

4. Cicli e condizioni

4.1 Cicli

Obiettivo di apprendimento

- ☐ Conosco i vari tipi di cicli e so quando e come utilizzarli.

Nota bene

La regola d'oro della programmazione è:

Non scrivere due volte una cosa
se puoi scriverla una volta sola!

4.1.1 Ciclo con un contatore

Il tipo più semplice di ciclo è quello in cui si indica quante volte deve accadere qualcosa.

Esempio: il tuo robot deve percorrere un quadrato il cui lato misura esattamente 17,5 cm (è la circonferenza delle ruote standard SPIKE, per cui puoi indicare esattamente una rotazione del motore = 360 gradi). Sulla matrice luce si deve ora visualizzare il numero di lati. Il codice si presenta così: poiché sai che un quadrato ha quattro lati uguali e quattro angoli retti, puoi farlo con un ciclo in cui un lato e un angolo si ripetono quattro volte.

```
1  from hub import light_matrix
2  import motor_pair
3  from hub import port, sound
4  import runloop
5
6  async def main():
7
8      motor_pair.pair(motor_pair.PAIR_1, port.A, port.B)
9
10     i = 0
11     while i < 4:
12         light_matrix.write(str(i + 1))
13         await motor_pair.move_for_degrees(motor_pair.PAIR_1, 360, 0, velocity=360)
14         await motor_pair.move_for_degrees(motor_pair.PAIR_1, 210, 100, velocity=180)
15         i += 1
16     sound.beep(440, 500, 100)
17
18 runloop.run(main())
```

Osserviamo più attentamente questo codice. Conosci già le righe import, perché le abbiamo già usate negli esempi precedenti. Alla riga 3 si importano due elementi dalla sezione **hub.**

Riga 8: accoppiamo i motori di azionamento alle porte A e B.

Riga 10: impostiamo un **contatore** e il valore di avvio che ha questo contatore prima dell'inizio del ciclo. Questo spesso è denominato **`i`** oppure con un'altra singola lettera dell'alfabeto. Il contatore è una variabile. Più avanti esamineremo più attentamente le variabili. Come valore di avvio per il contatore si indica in genere `0`.

Suggerimento
La denominazione **`i`** di questi contatori risale ancora agli inizi della programmazione, quando si cercava di ridurre tutto al minimo, perché lo spazio di memorizzazione era prezioso. È un'abbreviazione del termine inglese

«integer», che significa «numero intero», cioè un numero come 0, 1, 256 o -13. Infatti, di solito si conta in numeri interi, non in frazioni.

Se, come in questo esempio, hai un solo contatore, puoi utilizzare la **i.** Se però devi contare più cose, si fa presto a sbagliare. Alcuni programmatori semplicemente utilizzano per il secondo contatore del programma la lettera **j**, ma col tempo diventa poco chiaro! Perciò conviene trovare un nome facile da ricordare. Nell'esempio possiamo chiamare il contatore **NUMERO_LATO.** Il codice si presenta quindi così:

```
NUMERO_LATO = 0
while NUMERO_LATO < 4
    # Qui c'è il codice per il lato e la rotazione
    NUMERO_LATO += 1
```

Riga 11: qui diciamo con quale frequenza deve svolgersi il ciclo. Nel nostro esempio vogliamo percorrere un quadrato, perciò il ciclo si svolgerà quattro volte. Se, come prevede la regola, il valore di avvio del contatore è `0`, dobbiamo contare fino a `3` (`0`, `1`, `2`, `3`). Non è del tutto logico nella vita reale, ma nella programmazione si fa generalmente così. Nel codice lo esprimiamo così:

`while i < 4`, oppure in italiano: finché **i** è inferiore a `4`.

Righe 12-15: qui c'è il codice che deve essere ripetuto nel ciclo. Per vedere esattamente cosa c'è nel ciclo, indentiamo questo codice. Dopo si procede senza indentare il codice.

Suggerimento

Come già sai da Scratch, quello che va ripetuto si trova nella parentesi del blocco C. Dopo si procede con qualcos'altro.

Riga 14: il parametro con il numero (nell'esempio di codice `210`) dipende da come è costruito il robot. Più le ruote sono vicine e più piccolo è il numero, più sono distanti e più grande è il numero. Anche la grandezza delle ruote influisce. Qui devi fare un po' di esperimenti. Ti consigliamo di provare!

Riga 15: `i += 1`
Alla fine incrementiamo il contatore di uno. Di conseguenza varia il valore di **i.**

Questo strano modo di scrivere è un'abbreviazione di
`i = i + 1`
e significa per esteso: il contatore i ora deve avere il valore del contatore precedente più 1. Ossia: se prima **i** era 2, adesso diventa 3.

Riga 16: il ciclo è terminato. Per controllare, emettiamo un breve bip. Questo codice non è più indentato, perché non fa più parte del ciclo.

Compito 4.1.1

- Modifica il codice per il quadrato in modo che dopo ogni rotazione venga emesso un bip. Una volta concluso il quadrato, sulla matrice luce deve apparire per un secondo un quadrato (soluzione alla fine della guida).

4.1.2 Ciclo continuo
4.1.3 Ciclo con una condizione di arresto

4.2 Condizioni

Obiettivo di apprendimento

- ☐ Sono in grado di utilizzare correttamente condizioni sia semplici che nidificate.

4.2.1 Condizioni semplici

In una condizione confrontiamo nuovamente due cose tra loro, ad esempio un valore del sensore con un valore prescritto. Per effettuare il confronto ci serviamo di nuovo del doppio segno di uguale **==.**

Lo facciamo con un sensore di distanza collegato alla **porta D**.

Esempio: dopo che il robot ha effettuato un determinato percorso, deve misurare quanto dista un ostacolo (ad esempio la parete, la cassetta gialla SPIKE o la tua mano). Dovrà emettere un suono diverso a seconda della distanza.

```
1  import motor_pair
2  from hub import port, sound
3  import distance_sensor
4  import runloop
5
6  async def main():
7
8      MP = motor_pair.PAIR_1
9      DISTANZA = distance_sensor.distance(port.D)
10
11     motor_pair.pair(MP, port.A, port.B)
12     await motor_pair.move_for_degrees(MP, 320, 0, velocity = 360)
13
14     if DISTANZA < 100:
15         sound.beep(330, 500, 100)
16     elif DISTANZA< 250:
17         sound.beep(440, 500, 100)
18     else:
19         sound.beep(660, 500, 100)
20
21 runloop.run(main())
```

Esaminiamo il codice con più attenzione.

Righe 8 e 9: qui introduciamo per la prima volta i nomi abbreviati, che ci aiutano a semplificare un po' il codice e a renderlo anche più comprensibile per noi.

Avrai certamente notato nel capitolo sulle coppie di motori che le righe di codice spesso sono molto lunghe e che espressioni come **`motor_pair.PAIR_1`** appaiono molto frequentemente. Lo stesso

accade con `distance_sensor.distance(port.D)` ogni volta che abbiamo a che fare con le distanze.

Qui abbiamo leggermente semplificato queste parti di codice complicate. In futuro, in questo programma potremo sempre scrivere `CM` (da «coppia di motori») quando di fatto intendiamo `motor_pair.PAIR_1`, mentre `DISTANZA` quando ci riferiamo al `distance_sensor.distance(port.D)` .

Suggerimento

È buona norma abituarsi a usare sempre lo stesso stile di scrittura per questi nomi abbreviati (che dal punto di vista tecnico sono delle variabili). Ad esempio potresti scrivere questi nomi abbreviati sempre in **italiano**, perché le normali parole del codice sono in inglese. Così potrai distinguerle più facilmente. Oppure, come in questo esempio, puoi scrivere i nomi abbreviati in **MAIUSCOLO**. Fai come ti sembra più comodo, basta che sia un sistema uniforme. Ti consente di mantenere facilmente il controllo anche con un codice lungo.

Righe 14-19: adesso questa è la condizione. Come vedi, è composta da tre parti.

Per prima cosa facciamo una verifica. La distanza è inferiore a 10 cm?
```
if DISTANZA < 100:
```
(Ricorda che il sensore di distanza misura in millimetri!) Poi l'hub deve emettere un suono bip profondo. Indentiamo questa azione.

Se non è inferiore a 10 cm, passa a un'altra richiesta. La distanza è inferiore a 25 cm?
```
elif DISTANZA < 250:
```
Segue un bip di media intensità. Adesso potremmo fare ancora tante altre misurazioni, fino a raggiungere una distanza di 2 metri, cioè la massima distanza che il sensore di distanza è in grado di riconoscere. Il codice sarebbe sempre uguale, inizierebbe con `elif`. `elif` è una contrazione delle parole inglesi «else» e «if» e significa: se la condizione sopra è `False`, verifica in base a questo secondo criterio.

Se anche da questa seconda verifica risulta False, segue un'ultima istruzione. Se tutte le misurazioni precedenti danno False, emetti un bip di forte intensità.
```
else:
    sound.beep(660, 500, 100)
```

Suggerimento

Affinché il risultato sia chiaro, rispetta sempre l'ordine delle richieste. Esaminiamo ora un esempio volutamente sbagliato.

```
14      if DISTANZA < 250:
15          sound.beep(330, 500, 100)
16      elif DISTANZA< 100:
17          sound.beep(440, 500, 100)
18      else:
19          sound.beep(660, 500, 100)
```

Non funzionerebbe, perché la richiesta alla riga 16
```
      elif DISTANZA< 100:
```

viene eseguita solo se la richiesta alla riga 14 dà il risultato `False`! Se la distanza non è inferiore a 25 cm, di certo non può essere inferiore a 10 cm!

4.2.2 Condizioni nidificate

5. Funzioni

Obiettivo di apprendimento

- ☐ So cos'è una funzione e per cosa è utile.
- ☐ Sono in grado di creare e richiamare funzioni con e senza parametri.
- ☐ Sono in grado di creare una funzione con un valore di ritorno e accedere a questo valore.

4.2.3 Funzioni senza parametri

Ti ricordi come abbiamo imparato la riga di codice **async** `def main()`? La regola è: «Un programma complesso è spesso suddiviso in più parti, definite funzioni». Adesso è arrivato il momento di imparare a conoscere meglio le funzioni.

Ti ricordi ancora la regola d'oro della programmazione? Eccola di nuovo:

Nota bene

La regola d'oro della programmazione è:

Non scrivere due volte una cosa
se puoi scriverla una volta sola!

Quando si esegue un'azione (o un gruppo di azioni) più volte consecutivamente è opportuno ricorrere a un ciclo. Cosa si fa, però, se tra i due passaggi di questa azione succede qualcos'altro che non viene ripetuto? Bisogna scrivere il codice due volte, con tutti gli svantaggi che questo comporta?

No, proprio per questo esistono le **funzioni**. Con una funzione si esternalizza una parte di programma per poterlo in seguito richiamare con una sola riga di codice ogni volta che lo si desidera.

Esempio: il robot deve percorrere un quadrato. Dopo ogni angolo deve eseguire un'azione diversa, ad es. mostrare un testo diverso sulla matrice luce oppure emettere un suono diverso.

```
1  from hub import port, sound, light_matrix
2  import motor_pair, runloop, time
3
4  CP = motor_pair.PAIR_1
5  motor_pair.pair(CP, port.A, port.B)
6
7  async def Percorri_un_lato():
8      await motor_pair.move_for_degrees(MP, 360, 0)
9      await motor_pair.move_for_degrees(MP, 210, 100)
10
11 async def main():
12     await Percorri_un_lato()
13     await sound.beep(440, 500, 100)
14     await Percorri_un_lato()
15     light_matrix.show_image(light_matrix.IMAGE_HAPPY)
16     time.sleep_ms(1000)
17     light_matrix.clear()
18     await Percorri_un_lato(MP)
```

```
19      await sound.beep(500, 500, 100)
20      await Percorri_un_lato(MP)
21      print("Fine!")
22  runloop.run(main())
23
24  raise SystemExit
```

Righe 4-5: per prima cosa definiamo i nomi abbreviati e le variabili occorrenti nell'intero programma. Nel nostro esempio il nome abbreviato **`CP`** sta per la coppia di motori con cui il robot si muove. Impostiamo anche il relativo valore. È **`motor_pair.PAIR_1.`** Qualunque sia il punto del codice in cui vogliamo dare il comando di avvio, con il nome abbreviato **`CP`** possiamo accedere a questa coppia di motori.

Si chiama **variabile globale** e viene definita **all'esterno** delle singole funzioni. Se invece crei una variabile **all'interno** di una funzione, la puoi usare solo in questa funzione.

Righe 7-9: qui definiamo la nostra funzione **`Percorri_un_lato()`**. Poiché in questa funzione ci occorrono comandi «awaitable», all'inizio ci deve essere la parola chiave **`async`**, come anche in **`main()`** . Il codice all'interno della funzione lo conosci già dal ciclo.

Righe 12, 14, 18 e 29: qui la nostra funzione viene richiamata quattro volte. Come vedi, ti serve solo un'unica riga per tutto quello che c'è in una funzione, qualunque sia la lunghezza effettiva della funzione.

Prima di richiamare la funzione è importante inserire la parola chiave **`await`**, affinché la funzione venga eseguita completamente prima della riga di codice successiva.

4.2.4 Funzioni con parametri
4.2.5 Funzioni con valore di ritorno

Esistono anche funzioni con un valore di ritorno. Vale a dire che non contengono (o non contengono solo) un'azione ma hanno un risultato che viene rimandato a **`main()`**, dove può essere riutilizzato.

Esempio: sicuramente ricordi il calcolo complicato con cui hai convertito la lunghezza del percorso che vuoi compiere in gradi di rotazione dei motori:

$$\frac{45\ cm}{17{,}5\ cm} = 2.571$$

E come seconda operazione:

$$2.571 * 360° = 925.56°$$

Adesso vogliamo scrivere una funzione che esegua automaticamente questo calcolo.

Come premessa convertiamo il calcolo in una formula di cui utilizziamo le parti come variabili.

$$\frac{PERCORSO\_IN\_CM}{CIRCONFERENZA\_RUOTA\_IN\_CM} = numero\ 1$$

Il numero 1 è un risultato intermedio che usiamo nella seconda formula, ovvero la seguente:

$$Numero\ 1 * 360° = GRADI$$

Possiamo sintetizzarlo in una formula:

$$GRADI = 360° * \frac{PERCORSO\_IN\_CM}{CIRCONFERENZA\_IN\_CM}$$

E sappiamo che la circonferenza delle ruote standard SPIKE è di 17,5 cm. Significa che nel codice, al posto della variabile **CIRCONFERENZA_IN_CM**, possiamo usare direttamente questo valore oppure all'inizio del codice abbiamo una variabile globale alla quale assegniamo questo valore. Questo procedimento ha il vantaggio che, in caso di successiva modifica del robot con altre ruote, ci basta usare il codice e modificare solo questa riga.

Vediamo ora come possiamo scrivere la funzione.

```
1  from hub import port
2  import motor_pair, runloop
3
4  CP = motor_pair.PAIR_1
5  motor_pair.pair(CP, port.A, port.B)
6  CIRCONFERENZA_IN_CM = 17,5  # Questa è una variabile globale!
7
8  def Calcola_gradi_da_cm(PERCORSO_IN_CM):
9      GRADI = 360 * PERCORSO_IN_CM / CIRCONFERENZA_IN_CM
10     GRADI = round(GRADI)
11     print(GRADI)
12     return GRADI
13
14 async def main():
15     motor_pair.move_for_degrees(MP, Calcola_gradi_da_cm (45), 0)
16
17 runloop.run(main())
```

Righe 4-6: definiamo variabili globali, affinché il codice resti ben chiaro dopo.

Righe 8-12: questa è la funzione. Ha un parametro **PERCORSO_IN_CM** che riceve da **main()**.

Riga 9: ecco come si presenta nel codice la precedente formula.

Nota bene

Nel codice puoi fare qualsiasi calcolo che faresti anche in una lezione di matematica. Solo alcune cose vengono scritte un po' diversamente. Questi sono gli operatori principali (segni matematici):
+ più
- meno
***** per
/ diviso
^ elevato alla potenza (ci serve perché nel codice Python i numeri non si possono scrivere come esponente. 3^2 nel codice si scrive così: **3^2**)

Riga 10: ora arrotondiamo il risultato al prossimo numero intero, perché ovunque ci servano dei gradi abbiamo bisogno di numeri interi. I numeri decimali generano un errore.

Alla riga 9 abbiamo detto che la variabile **GRADI** deve avere il valore calcolato dalla formula. Con 45 cm sarà 925,6. Alla fine della riga 9 ora è il nuovo valore di **GRADI.**

Alla riga 10 modifichiamo nuovamente il valore delle variabili e arrotondiamo l'ultimo valore noto.

Avvertenza

Il codice delle righe 9 e 10 non è un'equazione matematica! L'insegnante di matematica non sarebbe affatto entusiasta se risolvessi così un'equazione, perché a sinistra e a destra del segno d'uguaglianza non c'è la stessa cosa.

Dipende dal fatto che qui non abbiamo un'equazione bensì una variabile a cui abbiamo attribuito un valore. E questo valore può cambiare da una riga all'altra nel codice!

Abbiamo già visto un esempio nei cicli, dove abbiamo programmato un quadrato utilizzando un contatore (**`i`** o **`NUMERO_LATO`**). Abbiamo modificato anche questo contatore quando lo abbiamo incrementato di uno.

Riga 11: è di fatto una riga di aiuto, con cui scriviamo il valore dei GRADI nella console. Così possiamo verificare se tutto è giusto. Questa riga si può successivamente cancellare senza problemi.

Riga 12: qui si stabilisce che i **`GRADI`** sono il valore di uscita che la funzione restituisce a **`main()`**. Per questo ci serve la parola chiave **`return`**.

Riga 15: in questa riga si impiega la nostra funzione. Conosciamo già il comando **`motor_pair.move_for_degrees(`*`coppia_di_motori, gradi, sterzata`*`)`**.
Per la ***`coppia_di_motori`*** utilizziamo la sigla **`CP`**, definita all'inizio del codice alla riga 4.

- Per ottenere i ***`gradi`*** richiamiamo ora la funzione **`Calcola_gradi_da_cm`** con il parametro `45`, perché vogliamo percorrere un tratto di 45 cm. Quando il programma viene eseguito entra nella funzione, calcola il risultato e imposta il valore di ritorno della funzione come ***`gradi`***, cioè 926.
- Poiché vogliamo procedere diritto, ***`sterzata`*** è pari a `0`.

Soluzioni

In questa sezione trovi le soluzioni ai compiti in cui devi scrivere un codice.

Soluzione della parte 2.2 Due motori
Soluzione della parte 4.1.1. Cicli