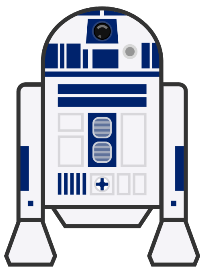


Sphero R2D2 Swift Playground Command Reference

Roll

```
roll(heading: Int, speed: Int)
```
heading: 0 - 360
speed: 80 - 255

Dependencies:
Requires wait() followed by stopRoll() as subsequent commands

```
wait(for: Double)
stopRoll() // stops roll action
```

Stance

```
setStance(R2D2Stance. [bipod | tripod | waddle | stop ])
```

Dependencies:
requires `wait(for: 3)` as next command after `R2D2Stance.waddle`.

```
wait(for: 3.0)
```

Sound

```
playSound(sound: R2D2Sound. [happy | cautious | excited | hello | joyful
| sad | scan | scared | talking])
```

Dome

```
setDomePosition(angle: Int)
```
angle: -100 - 100

Dependencies:
requires „wait(for: Int)“ command as next command

```
wait(for: Double)
```

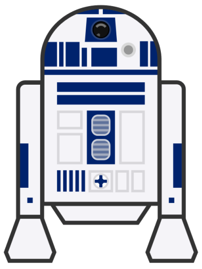
Lights

```
setFrontPSILed(color: FrontPSIColor. [black | blue | red])
setBackPSILed(color: BackPSIColor. [black | green | yellow])
```

```
setHoloProjectorLed(brightness: Int)
setLogicDisplayLeds(brightness: Int)
```
brigntness: 0 - 255

Wait

```
wait(for: Int)
```
for: 0 - 255



Sphero R2D2 Swift Playground Tutorial

Preface

Limiting your creativity

Swift Playgrounds sometimes limits you to a subset of the possible solutions to a problem. It can happen that you are able to program syntactically correct answer but R2D2 just shakes it's head and makes a beeping noise. Your solution in a formal way might be correct but the person who programmed the playground limited your freedom for writing the code you want.

For example on Page 3 - Head Swivel this would be formally correct code but it fails as the developer requested you the set the angle to -100 or 100 degrees - and nothing else. Always keep in mind that even if your code is correct it might not be what the playground expects.

```
setDomePosition(angle: -100)
wait(for: 1.0)
setDomePosition(angle: 100)
wait(for: 1.0)
setDomePosition(angle: 0)
play(sound:R2D2Sound.cautious)
```

Paste the same code on a different page and R2D2 will swivel it's dome in any angle you specify.

This is not because of a bad programmer or a bug, but to make you stick to the task and limit distraction.

R2D2 Playground exercises

Page 2: Roll

In this lecture you are tasked with driving R2D2 along a course.
To fulfill this task you‘ll need the following commands, which are described in detail above in the command reference:

- setStance()
- roll()
- wait
- play()

Here are some examples of how to concour the task. From a basic linear approach to a solution that uses a array of structs that describe the path that R2D2 will drive along.

2.1 Basic version

```
func escape() {
    setStance(R2D2Stance.tripod)
    roll(heading: 0, speed:80)
    wait(for: 1.0)
    stopRoll()
    play(sound: R2D2Sound.happy)
}
```

Explanation
This code snippet lets R2D2 drive for one second on a straight line. At the beginning he sets the stance and polls it in after he arrived. Then he plays a happy sound.

Task
Play along with the values of R2D2Stance, heading, speed, for and R2D2Sound in order to understand the basics of what R2D2 is capable of.

2.2 Using an array for patrolling in a square

```
func escape() {
    let angle = [0, 90, 180, 270]
    setStance(R2D2Stance.tripod )
    for a in angle {
        roll(heading: a, speed:80)
        wait(for: 1)
    }
    stopRoll()
    play(sound: R2D2Sound.happy)
}
```

Explanation
This code snippet lets R2D2 drive along a path that is defined by an array which contains the heading values. In a for loop the value for heading is read from the array.

Task
Play along with the angle array. Find other values for the angle array. Add more values to the array.

2.3 Using arrays for angle and speed

```
func escape() {
    let angle = [0, 90, 180, 270]
    let speed = [80, 100, 120, 140]
    setStance(R2D2Stance.tripod )
    for i in 0 ... 3 {
        roll(heading: angle[i], speed: speed[i])
        wait(for: 2.0)
    }
    stopRoll()
    play(sound: R2D2Sound.happy)
}
```

Good to know:
Arrays always start at index 0.

Explanation
This code snippet lets R2D2 drive along a path that is defined by an array which contains the heading values. The corresponding speed is defined in a second array named speed. In a for loop the values for heading and speed are read from the arrays.

Task
Play along with the angle and speed arrays. Find other values for the angle and speed array. Add more values to the arrays.

Quiz
- What happens, if you add more values to the angle array but not to the speed array?
- How could you prevent this? *
- Which other value do you have to change if you add more values to the arrays - why?

*compare angle.count to speed.count and loop to the smaller value

2.4 Using a two dimensional array for angle and speed

```
func escape() {
    let angle = 0
    let speed = 1
    var way = [[0,80], [90,100], [180,120], [270,140], [45,255]]
    setStance(R2D2Stance.tripod )
    for waypoint in way {
        roll(heading: waypoint[angle], speed: waypoint[speed])
        wait(for: 2.0)
    }
    stopRoll()
    play(sound: R2D2Sound.happy)
}
```

Explanation
This code snippet lets R2D2 drive along a path that is defined by an 2-dimensional array which contains the heading and speed values. The constants angle and speed are indexes into the array. This gives us better readability of the code. In a for loop the values for heading and speed are read from the array.

Task
Play along with the angle array. Find other values for the angle array. Add more values to the array.

Quiz
- What is the advantage of the 2-dimensional array?*

*Heading and speed are defined in pairs as required.

2.5 Using a function to initialize the two dimensional array

```
func makeWay(numberOfWayPoints: Int) -> [[Int]] {
    let angle = 0
    let speed = 1
    var way = Array(repeating: Array(repeating: 0, count: 2), count:
numberOfWayPoints)
    var angleValue = 0
    var speedValue = 80
    for i in 0 ... numberOfWayPoints-1 {
        way[i][angle] = angleValue
        way[i][speed] = speedValue
        angleValue += Int(360/numberOfWayPoints)
        speedValue += Int(175/numberOfWayPoints)
    }
    return way
}

func escape() {
    let angle = 0
    let speed = 1
    var way = makeWay(numberOfWayPoints: 6)

    setStance(R2D2Stance.tripod )
    for waypoint in way {
        roll(heading: waypoint[angle], speed: waypoint[speed])
        wait(for: 1.0)
    }
    stopRoll()
    play(sound: R2D2Sound.happy)
}
```

Explanation
Here we use a function to create and initialize the 2-dimensional array. When the array way is created, it is initialized with zeros.
The variable angleValue is calculated as a fraction of a full circle (360°). The variable speedValue is calculated as a fraction of (maximum speed - minimum speed) with an offset of minimum speed. With this we always do a full circle in our movement and we give R2D2 linear acceleration.
This time we loop over the array using an index variable i.

Task
Find other ways to fill the array distributing angle and acceleration differently.
Rewrite the code in a way it doesn't need i as an index variable.

Quiz
- Do you have to initialize the array immediately when the var is defined? *
- Why is using an index variable like i error prone?**

*???
** An Arrays index always starts at 0. Thus i will be out of bounds if you forget to subtract 1 from the number of elements in the array calculating the upper bound.

2.6 Using an array of struct

```
struct leg {
    var angle: Int
    var speed: Int
    var wait: Double
}

func makeWay(numberOfWayPoints: Int) -> [leg] {
    var way = [leg]()
    var angleValue = 0
    var speedValue = 80
    var waitValue = 1.0
    var thisLeg = leg(angle: 0, speed: 0, wait: 0.0)

    for i in 0 ... numberOfWayPoints-1 {
        thisLeg.angle = angleValue
        thisLeg.speed = speedValue
        thisLeg.wait = round((waitValue + (Double(i)/
                             Double(numberOfWayPoints)))*10)/10
        angleValue += Int(360/numberOfWayPoints)
        speedValue += Int(175/numberOfWayPoints)
        way.append(thisLeg)
    }
    return way
}

func escape() {
    var way = makeWay(numberOfWayPoints: 6)

    setStance(R2D2Stance.tripod )
    for waypoint in way {
        roll(heading: waypoint.angle, speed: waypoint.speed)
        wait(for: waypoint.wait)
    }
    stopRoll()
    play(sound: R2D2Sound.happy)
}
```

Explanation
Now we use an array of struct to describe R2D2s way. Doing so we could predefine a quite complex way R2D2 can go to get to the escape pod. Each part of the way is named a leg, like in a relay. For each leg we can now predefine not only angle and speed but also the distance to travel (which is defined by the wait time).

Task
Preinitialize the array way with zero values for numberOfWayPoints at declaration.

Quiz
Describe how thisLeg.wait is calculated. What could be the idea behind this?
Wouldn’t it be more memory efficient to declare leg.wait as Float instead of Double - discuss this.

Page 3 - HeadSwivel

Actually there is not much to do in this lesson. It teaches you to set the dome position and that you have to wait for some amount of time before you can turn the dome to the opposite direction.

To fulfill this task you'll need the following commands, which are described in detail above in the command reference:

- setDomePosition()
- wait
- play()

3.1 Basic Version

```
func domeSwivel {
      setDomePosition(angle: -100)
      wait(for: 1.0)
      setDomePosition(angle: 100)
      wait(for: 1.0)
      play(sound:R2D2Sound.cautious)
}
```

3.2 Swivel Dome with increasing angle

As written in the preface this code can't be executed on page 3, as this playground page limits you to the angles of -100 or 100 respectively. But you can run this code by copying it to lets's say page 4.

```
func domeSwivel {
      var angle = 10
      while (angle <= 100) {
            setDomePosition(angle: -angle)
            wait(for: 1.0)
            setDomePosition(angle: angle)
            wait(for: 1.0)
            angle += 10
      }
}
```

Explanation
Here we use a while loop to iterate through values from 10 to 100 for the angle.

Task
- Make R2D2 swivel the dome for different angles on either side e.g 100°, 90°, 80°, … on the right and 10°, 20°, 30°, … on the left.
- Make R2D2 swivel the dome first on the right side in steps of 10° and repeat that on the left side.

Quiz
How far can R2D2 turn it's dome - try it out?
What happens if you commit the wait() command?

Page 4 - Waddle

On this page R2D2 should waddle for at least 3 seconds.

4.1 Basic Version

```
func waddle() {
      setStance(R2D2Stance.waddle )
      wait(for: 3)
      setStance(R2D2Stance.stop )
      play(sound:R2D2Sound.scared)
}
```

Explanation
This is the minimal command sequence the playground expects.

Task
- Try other values within the wait() command.
- Try setStance() commands in a different order.

Quiz
- If you have access to a Macintosh computer find the file within the R2D2 playground in which the 3 second wait is defined.*
  In order to locate the file you have to right click on the R2D2 by Sphere Playground file and choose Show Package Content from the menu.

*/R2-D2\ by\ Sphero.playgroundbook/Contents/Chapters/01-Basics.playgroundchapter/Pages/04-Waddle.playgroundpage/Sources/WaddleAssessmentController.swift

---

4.2 All Possible Stance Options

```
func waddle() {
      setStance(R2D2Stance.tripod)
      wait(for: 1.0)
      setStance(R2D2Stance.bipod)
      wait(for: 1.0)
      setStance(R2D2Stance.waddle )
      wait(for: 3.0)
      setStance(R2D2Stance.stop )
      play(sound:R2D2Sound.scared)
}
```

Explanation
Here we try out all setStance() commands.

Task
- Try different sequences of the setStance() commands.
- Put the setStance() commands in a loop. Let R2D2 do some exercises on two and three legs.

Quiz
- Where is the code you write stored within the R2D2 by Sphere playground?*

*/R2-D2\ by\ Sphero.playgroundbook/Edits/UserEdits.diffpack/Chapters/01-Basics.playgroundchapter/Pages/04-Waddle.playgroundpage/Contents.swift.delta

*In Swift Playgrounds on the iPad click the Tools menu (…) navigate to the file from there. But you can't see the content of the file.

4.3 Investigate Environment Before Giving a Warning

Before R2D2 gives his warning he first investigates the environment around him by driving a perfect circle.

```
func driveCircle() {
    var angle = 0
    setStance(R2D2Stance.tripod)
    while angle &lt;= 360 {
        roll(heading: angle, speed: 80)
        wait(for: 2.0)
        angle += 36
    }
    stopRoll()
    setStance(R2D2Stance.bipod)
}

func waddle() {
      driveCircle()
      setStance(R2D2Stance.waddle )
      wait(for: 3.0)
      setStance(R2D2Stance.stop )
      play(sound:R2D2Sound.scared)
}
```

<u>Explanation</u>
In driveCircle we need a variable for the direction that R2D2 drives. This is the variable angle. In order to drive R2D2 has to go on all three legs. In the while loop we increase the angle setting to set the new direction.

<u>Task</u>
- Make the circle R2D2 is driving wider.
- Make R2D2 drive a circle of circles, drawing some kind of a bloom.

<u>Quiz</u>
- What happens if you use setStance(R2D2Stance.stop) instead of stopRoll()?
- How can you make the circle more accurate?*

* You can achieve this by making the increment of angle smaller and decrease the value for wait at the same time.

Page 5 - Lights

The instructions at the beginning of this page are a little bit unspecific (as in most IT projects).

Let's make things a little bit clearer. This is what is required in pseudo code:

- Set front and back LED to any color
- Play the talking sound
- In a loop call the function flashLights().
- In the function flashLights() turn on the Logic LED and the Holo LED, then wait, then turn both LEDs off and wait again.

The color of the LEDs is limited in the R2D2 by Sphero Playground. Every LED can only be one of two colors or turned off. In the SpheroEDU App you can set them to any color by setting RGB values. But for the purpose of learning Swift two colors are totally sufficient.

Actually the Lights page drove me crazy at first as I couldn't find the right implementation for the flashLights() function. I even opened a support call at Sphero. In the end I figured it out myself. The struggle with this lesson was my motivation to write this guide. To make it easier for others to understand and to minimize frustration over the playground requirements that have nothing to do with Swift itself. My intention is to free up time to fight real Swift and algorithmic problems not artificial limitations of the playground.
In the quiz of 4.1 I asked you to find out where playground file are stored because in those you can find the playground code the limits you if you are stuck in a playground.

5.1 Basic Version

Here we have the functionality that is required as a minimum.

```
func flashLights() {
    setLogicDisplayLeds(brightness: 255)
    setHoloProjectorLed(brightness: 255)
    wait(for: 1.0)
    setHoloProjectorLed(brightness: 0)
    setLogicDisplayLeds(brightness: 0)
    wait(for: 1.0)
}

func startMessage() {
    setFrontPSILed(color: FrontPSIColor.blue)
    setBackPSILed(color: BackPSIColor.yellow)
    play(sound: R2D2Sound.talking)
    for index in 1 ... 5 {
        flashLights()
    }
}
```

<u>Explanation</u>
The code for flashing the LEDs is isolated in a function. In flashLights() just turns on the LEDs, waits for one second and turns the LEDs off again, then waiting an other second.
In startMessage the function flashLights() is called in a simple for loop.

<u>Task</u>

- Transform this code back to a less preferable „spaghetti code“ programming style. Move the code in flashLights() back into startMessage()
- Starting with the example code above move the setFront/BackPSILed() function calls into an other function.

<u>Quiz</u>
- Why are functions preferred over „spaghetti code“?

5.2 Change Intensity of the Logic and the Holo LED

Here the intensity of the holo projector LED isn’t constant but it gets increasingly brighter. As the number of iterations is increased the message takes longer to project even though the wait is increased.

```
let off=0
let on=255

func flashLights(_ index: Int,_ iterations: Int) {
    let delay=0.1
    setLogicDisplayLeds(brightness: (index*on/iterations))
    setHoloProjectorLed(brightness:(index*on/iterations))
    wait(for: delay)
    setHoloProjectorLed(brightness:off)
    setLogicDisplayLeds(brightness:off)
    wait(for: delay)
}

func startMessage() {
    let iterations=51
    setFrontPSILed(color: FrontPSIColor.blue)
    setBackPSILed(color: BackPSIColor.green)
    play(sound: R2D2Sound.talking)
    for index in 1 ... iterations {
        flashLights(index, iterations)
    }
}
```

<u>Explanation</u>
The intensity increase per round of the loop is calculated as the maximum intensity divided by the iterations (number of loops). So in the beginning we have the fraction of the maximum intensity divided by the iterations. Every round of the loop we increase the intensity by that fraction. In the end we have iteration*max/iterations which after shortening the fraction we have max as the final intensity.

<u>Task</u>
- Modify the code in a way that intensity increases nonlinear, e.g. square. The while loop might be suited better for this.

<u>Quiz</u>
- What do the „_“ in the declaration or the function flashLights() do?
- How would you declare flashLights() without the „_“?
- How would you call flashLights() without the „_“?

5.3 Send a Textmessage

Now we display the message to Obi-Wan Kenobi that princess Leia recorded into R2D2, when her ship was attacked by the Imperial troops.

```
func sendMessage(_ message: String) {
    let off=0
    let on=255
    let delay=0.1
    var intensity: Int

    for char in message.unicodeScalars {
        intensity = Int(char.value)*2
    setLogicDisplayLeds(brightness: on)
    setHoloProjectorLed(brightness: intensity)
    wait(for: delay)
    setHoloProjectorLed(brightness:off)
    setLogicDisplayLeds(brightness:off)
    wait(for: delay)
    }
}

func startMessage() {
    let helpMessage:String = "Help me, Obi-Wan Kenobi. You're my only
hope. General Kenobi. Years ago you served my father in the Clone Wars.
Now he begs you to help him in his struggle against the Empire. I regret
that I am unable to present my father's request to you in person, but my
ship has fallen under attack, and I'm afraid my mission to bring you to
Alderaan has failed. I have placed information vital to the survival of
the Rebellion into the memory systems of this R2 unit. My father will
know how to retrieve it. You must see this droid safely delivered to him
on Alderaan. This is our most desperate hour. Help me, Obi-Wan Kenobi.
You're my only hope."
    setFrontPSILed(color: FrontPSIColor.blue)
    setBackPSILed(color: BackPSIColor.green)
    play(sound: R2D2Sound.talking)
    sendMessage(helpMessage)
}
```

Explanation
The message princess Leia sends to Obi-Wan is given in a string. We typecast the characters in this string to their equivalent ASCII Code. This maps every character in that string to an Int value between 0 and 127. In order to utilize the full range of possible brightness levels, the ASCII value is multiplied by 2. This makes the LED flash brighter.
As required by the playground we have to turn the LogicDisplayLED in sync with the HoloProjectorLED.

Task
- Help R2D2 to even out it's energy consumption by flashing the LogicDisplayLEDs less bright if the HoloProjectorLED flashes bright. *
- Write an other function that uses the setLogicDisplayLeds and setHoloProjectorLed commands. Call this new function from within send message.
- Read: https://developer.apple.com/documentation/swift/string.unicodescalarview and explain what the difference between a unicode character and its unicodeScalar is.

* (255 - HoloProjectorLED brightness)

Quiz
- Write a function that prints out the ASCII characters of the integers from 0 to 127.
  @@@ You have to use a different Swift Playground for this.