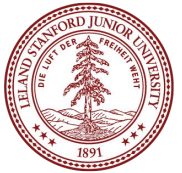
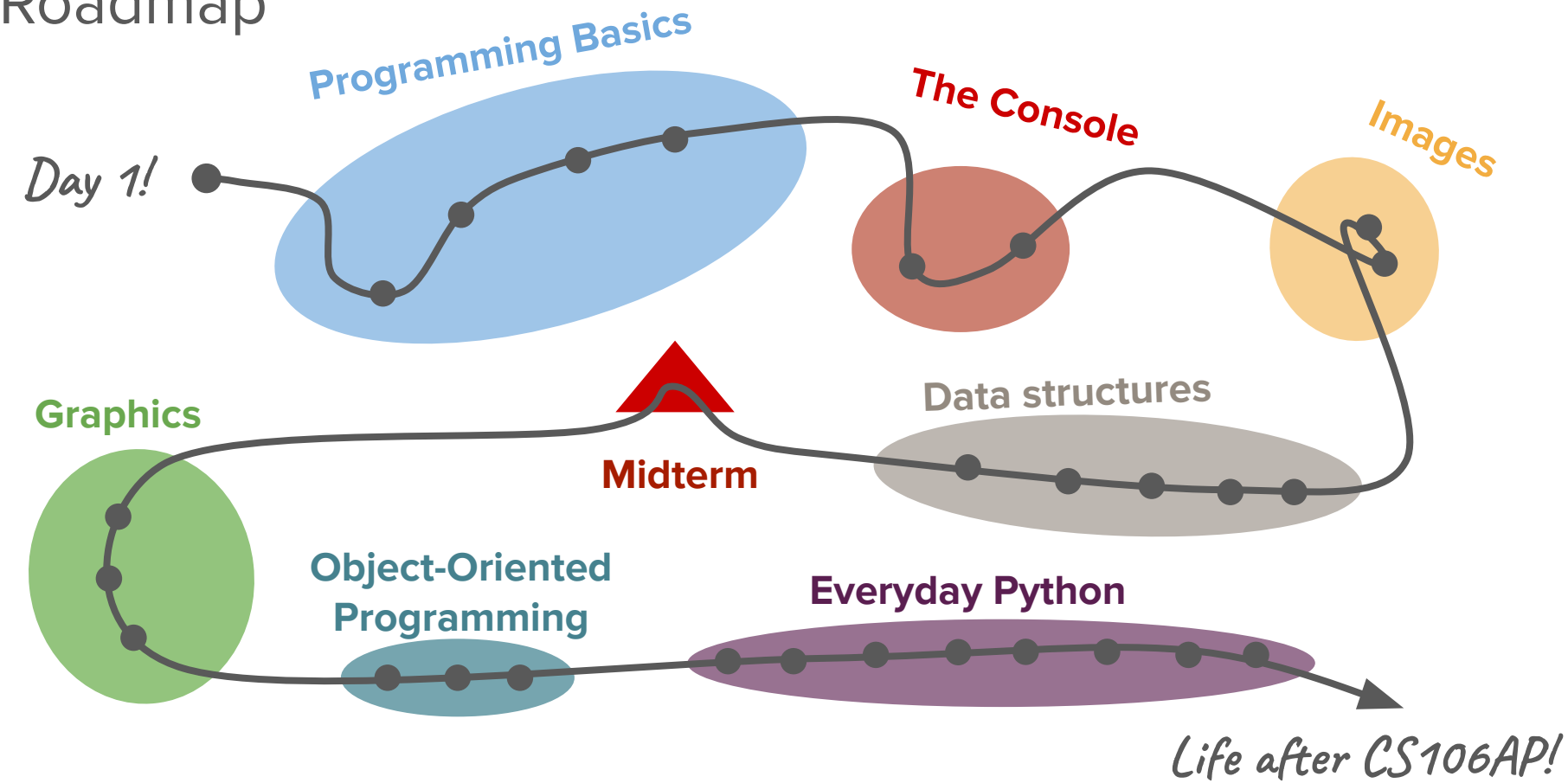
Style and (more) Control Flow

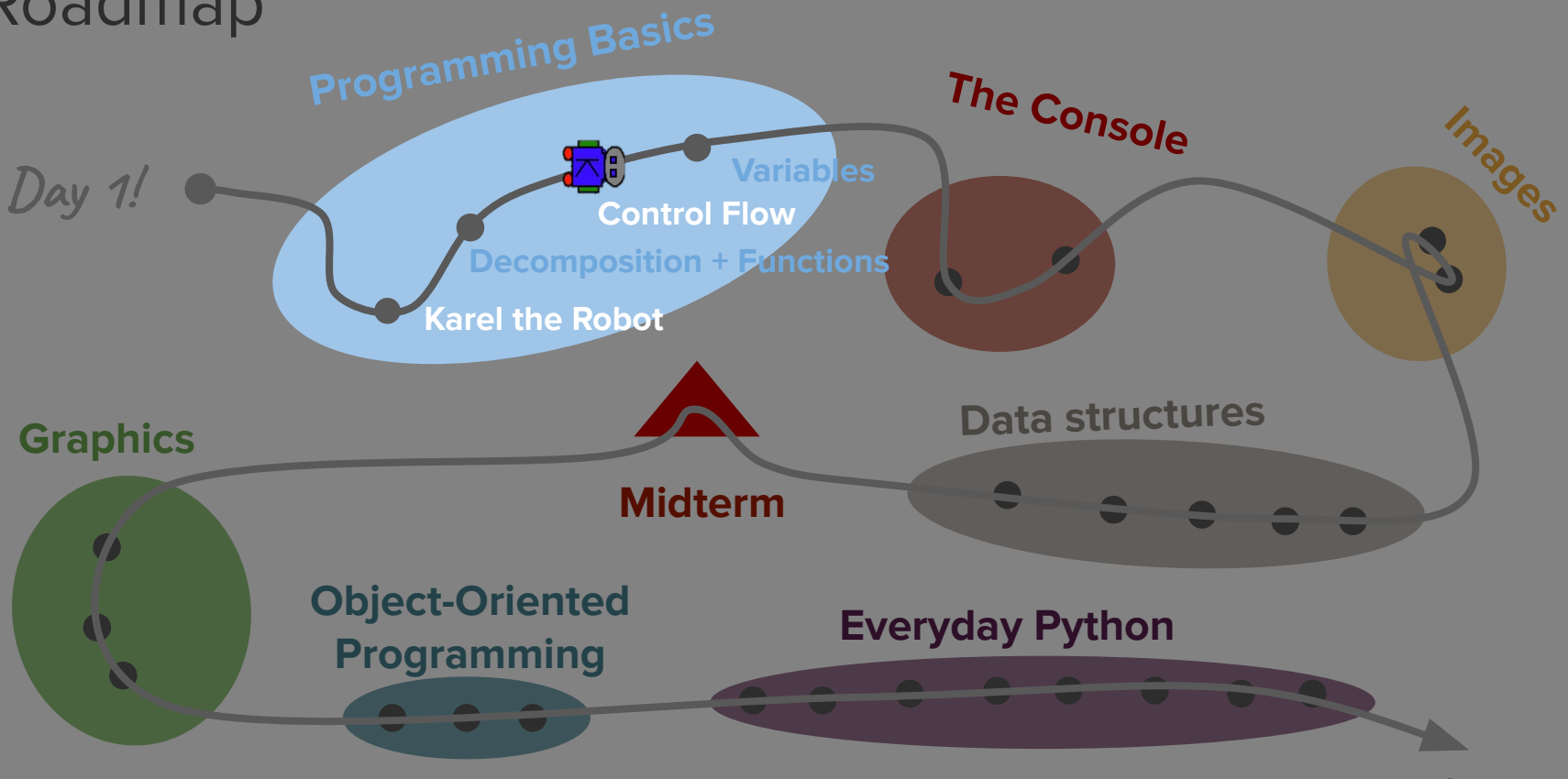
CS106AP Lecture 4



Roadmap


Programming Basics
Day 1!
The Console
Images
Graphics
Midterm
Data structures
Object-Oriented Programming
Everyday Python
Life after CS106AP!

Roadmap


Programming Basics
Day 1!
Variables
Control Flow
Decomposition + Functions
Karel the Robot
The Console
Images
Graphics
Midterm
Data structures
Object-Oriented Programming
Everyday Python
Life after CS106AP!

Today's questions

What do we mean by coding with "good style"?

How does control flow allow us to create more complex programs?

Today's topics

1. Review: HurdlesKarel
2. Coding style tactics
3. Control flow 2.0
   - **`and/or`**
   - Conditionals (if-else, if-elif-else)
   - Example: NinjaKarel
4. What's next?

Review

Program structure

The structure of a **program**

```
import ...


def main():
    ...


if __name__ == '__main__':
    ...
```

Definition

computer program

A set of instructions for the computer – an “app.” These instructions are also called lines of “source code.”

not

A note about **`not`**

- We can use **`not`** in front of a boolean expression to negate it!

```
if not boolean expression:

    # Do something if expression is not True
```

A note about **`not`**

- We can use **`not`** in front of a boolean expression to negate it!

```
if not front_is_clear():
    turn_around()
```

Top-down decomposition

Top-down decomposition

- “Divide-and-conquer”
  - Break the problem down into smaller parts
  - Ask: What are the steps that make up the larger problem? What tasks are repeated and might make good functions?

Top-down decomposition

- “Divide-and-conquer”

- Plan out your milestones (functions) first before writing them:
  - What are the pre-conditions and post-conditions for each function?
  - Which functions will call which?

Top-down decomposition

- “Divide-and-conquer”
- Plan out your milestones (functions) first before writing them
- Use the “blackbox model” for functions you’re not working on
  - When working on a particular function, assume that the others exist and already do what you want

Top-down decomposition

- “Divide-and-conquer”
- Plan out your milestones (functions) first before writing them
- Use the “blackbox model” for functions you’re not working on

Steps

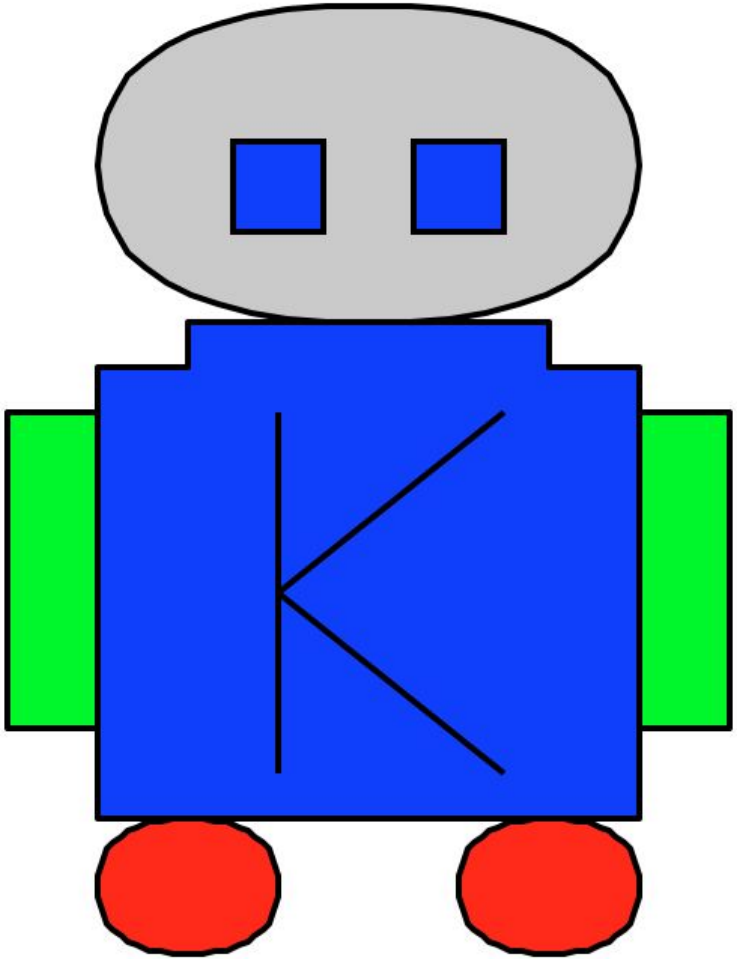
1. Plan your milestones (functions).

2. Write **all of the function definitions** and their pre-conditions and post-conditions in the function comments.

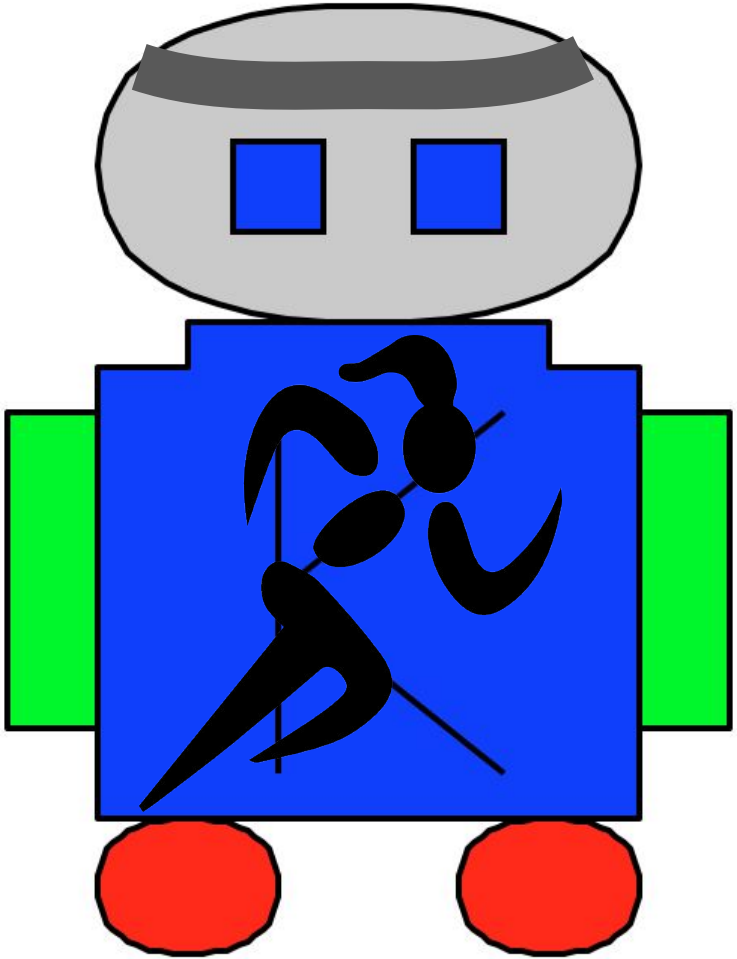
3. Work on **one function at a time**: write its code, test it, and debug as necessary before moving on to the next milestone (function).

4. Make sure your program works on all possible scenarios (worlds)!

Today we’re going to create...



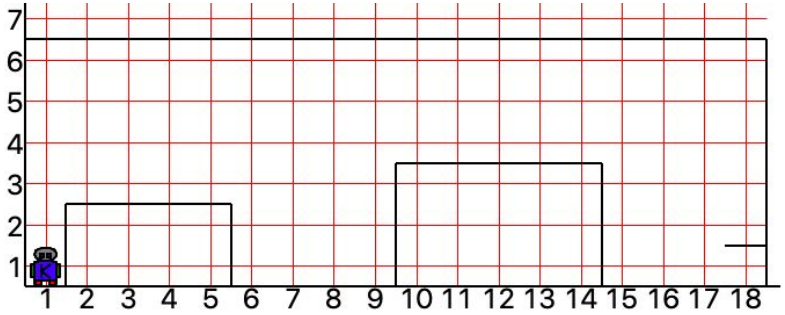
ATHLETE KAREL!



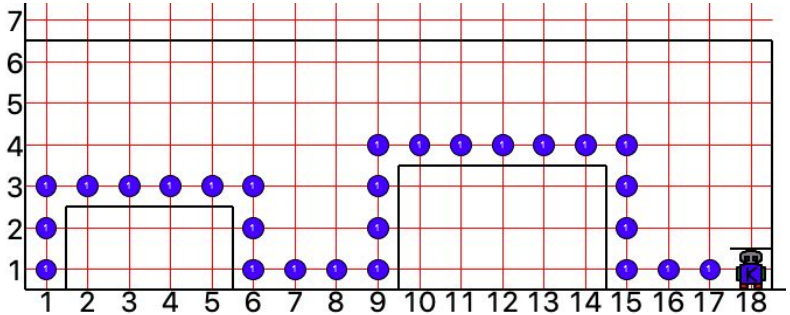
Another example:

`HurdlesKarel`

HurdlesKarel



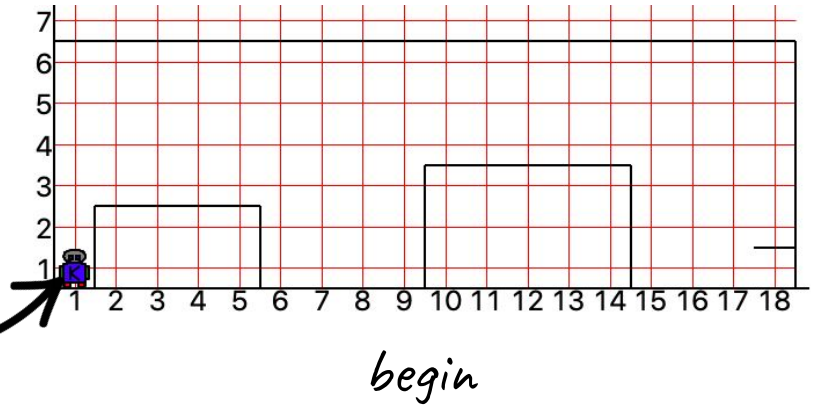
begin



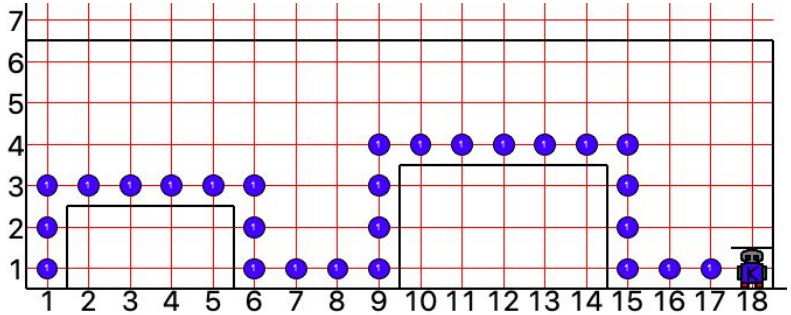
end

HurdlesKarel

Karel will always start with a hurdle directly to its right

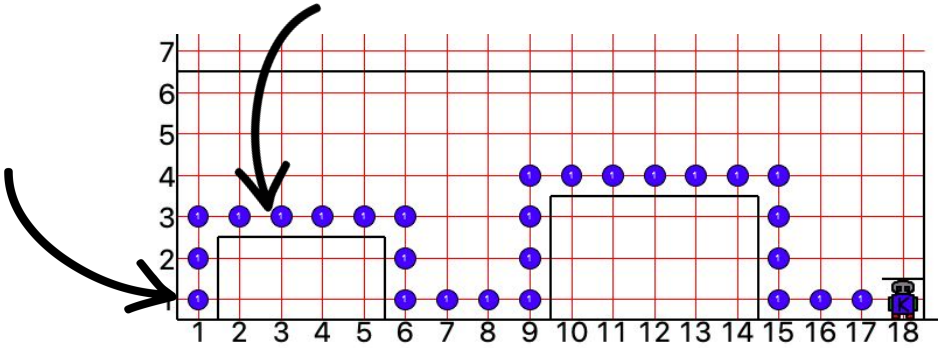


HurdlesKarel



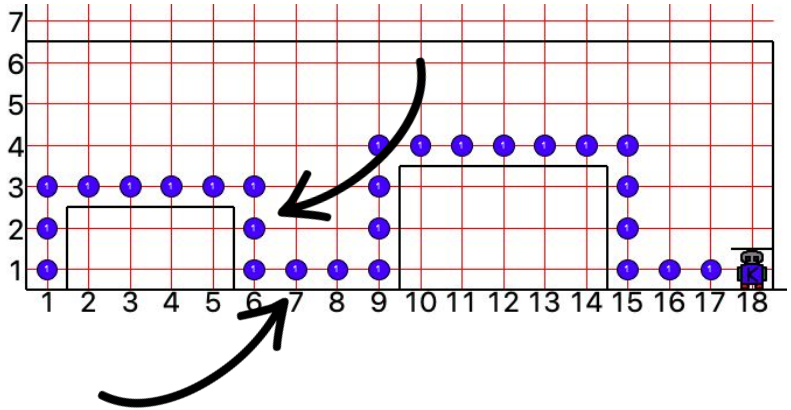
Four moves, two types

HurdlesKarel



Follow wall on right

HurdlesKarel



Follow wall until blocked

Steps

1. Plan your milestones (functions).

2. Write **all of the function definitions** and their pre-conditions and post-conditions in the function comments.

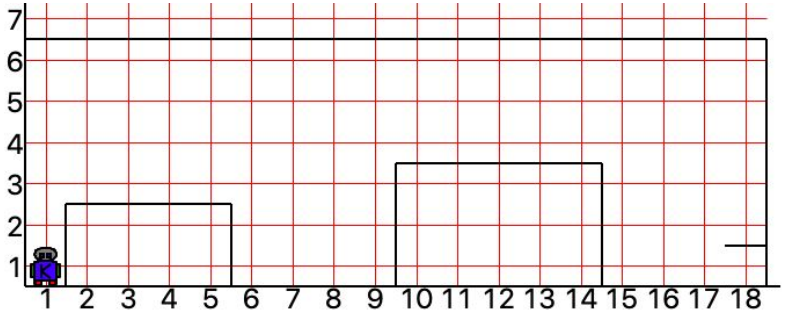
3. Work on **one function at a time**: write its code, test it, and debug as necessary before moving on to the next milestone (function).

4. Make sure your program works on all possible scenarios (worlds)!

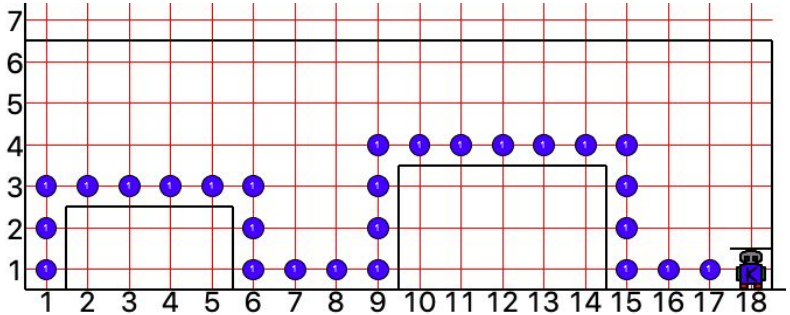
HurdlesKarel

[demo]

HurdlesKarel



begin



end

What do we mean by coding with “good style”?

In this class...

- Style pointers will be called out:

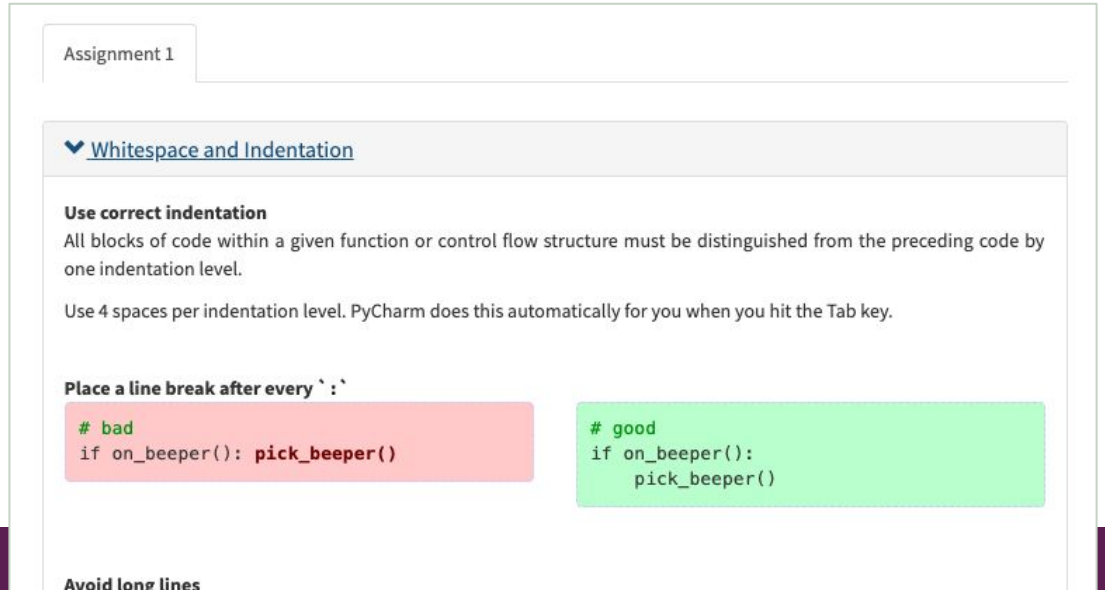
Style note

decomposition
Having multiple small functions makes your program easier to read and understand

In this class...

- Style pointers will be called out.

- Each assignment has a tab with tips in the course Style Guide.



In this class...

- Style pointers will be called out.
- Each assignment has a tab with tips in the course Style Guide.
- We will use PEP8!

A note about PEP 8

- PEP - Python Enhancement Proposal
- PEP 8 is an official standard for Python code formatting
- All the code we show you will follow PEP 8, and PyCharm also enforces it by default!
  - You should therefore follow this by default in your code.
- In real-world projects, some are super strict about following PEP 8.

What we've learned so far

- Have descriptive function names
  - Use verbs
  - Separate words with underscores (this is called "snake case")

What we've learned so far

- Have descriptive function names

- Decompose your programs
  - Use short functions to add descriptive names or add basic control flow lines to an existing command (e.g. **`move_safely()`**)
  - Create functions that reduce repeated code and isolate specific sub-problems.

What we've learned so far

- Have descriptive function names
- Decompose your programs
- Use consistent spacing (4 spaces per level of indentation!)
  - This is also a common source of bugs
  - Python uses indentation to understand your code (what's inside the function/loop/if statement)

What we've learned so far

- Have descriptive function names
- Decompose your programs
- Use consistent spacing (4 spaces per level of indentation!)
- Comment your code **as you write each function**
  - Include overall comments for every function
  - Use inline comments for complex blocks of code within functions

What we've learned so far

- Have descriptive function names
- Decompose your programs
- Use consistent spacing (4 spaces per level of indentation!)
- Comment your code **as you write each function**

How does control flow enable more complex programs?

Combining boolean expressions

Definition

boolean expression

A code snippet that evaluates to True or False

Used as the conditions in if statements and while loops!

Definition

boolean expression

A code snippet that evaluates to True or False

Karel functions that evaluate to boolean expressions

`front_is_clear()`	Is there no wall in front of Karel?
`left_is_clear()`	Is there no wall to Karel's left?
`right_is_clear()`	Is there no wall to Karel's right?
`on_beeper()`	Is there a beeper on the corner where Karel is standing?
`facing_north()`	Is Karel facing north?
`facing_south()`	Is Karel facing south?
`facing_east()`	Is Karel facing east?
`facing_west()`	Is Karel facing west?

`not`, `and`, & `or`

- Python uses the words **`not`**, **`and`**, & **`or`** to combine boolean values

not, **and**, & **or**

- Python uses the words **not**, **and**, & **or** to combine boolean values
  - **not** `boolean_expression`
    Inverts True/False

When is the condition true?

`a`	`not a`
true	false
false	true

not, **and**, & **or**

- Python uses the words **not**, **and**, & **or** to combine boolean values
  - **not** `boolean_expression`
    Inverts True/False

  - `boolean_expression_a` **and** `boolean_expression_b`
    If both sides are true, the entire condition is true.

When is the condition true?

`a`	`b`	`a and b`
true	true	true
true	false	false
false	true	false
false	false	false

`not`, `and`, & `or`

- Python uses the words **`not`**, **`and`**, & **`or`** to combine boolean values
  - `not boolean_expression`
    Inverts True/False
  - `boolean_expression_a and boolean_expression_b`
    If both sides are true, the entire condition is true.
  - `boolean_expression_a or boolean_expression_b`
    If either side is true, the entire condition is true.

When is the condition true?

`a`	`b`	`a or b`
true	true	true
true	false	true
false	true	true
false	false	false

not, **and**, & **or**

- Python uses the words **not**, **and**, & **or** to combine boolean values
  - **not** `boolean_expression`
    Inverts True/False

  - `boolean_expression_a` **and** `boolean_expression_b`
    If both sides are true, the entire condition is true.

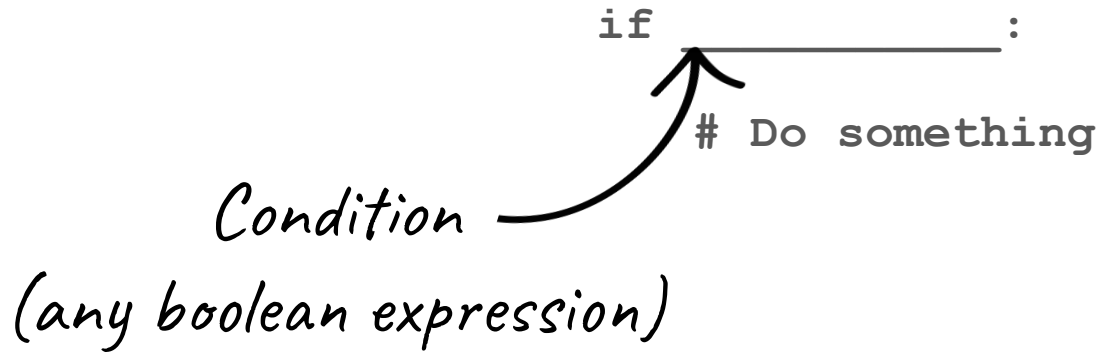
  - `boolean_expression_a` **or** `boolean_expression_b`
    If either side is true, the entire condition is true.

Conditionals (if-else)

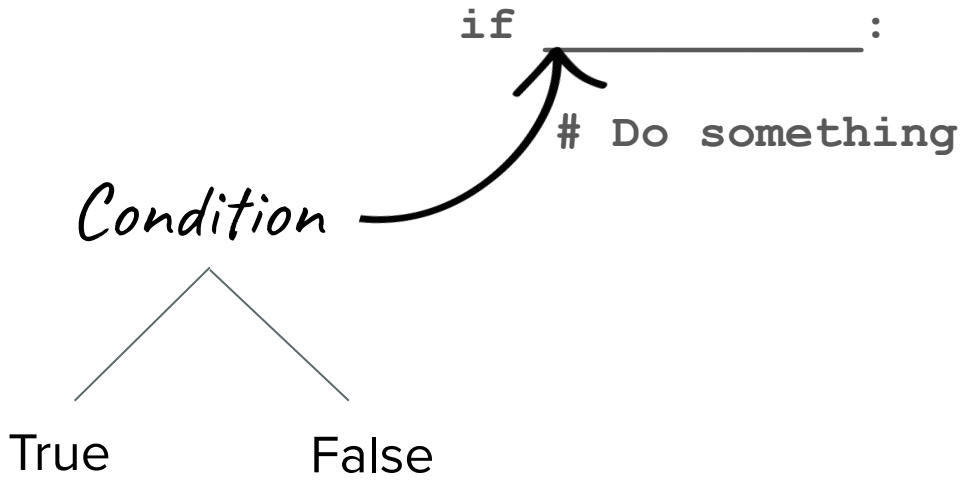
If statements

```
if _____________:

    # Do something
```

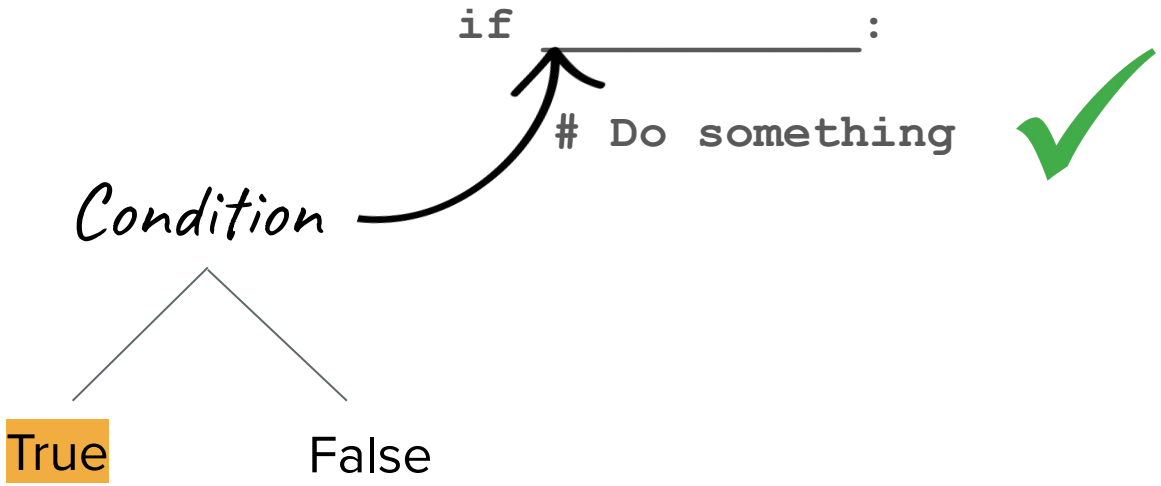
If statements



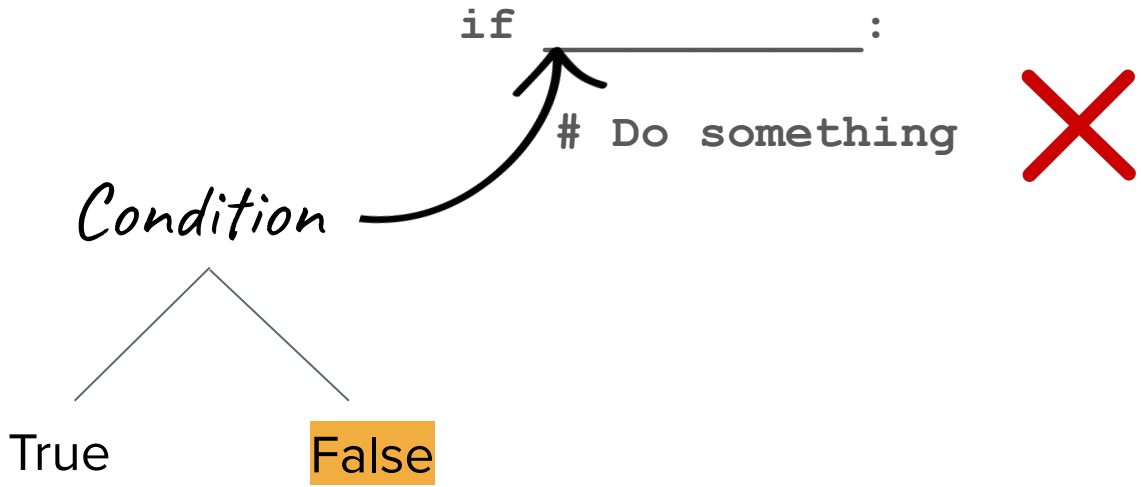
If statements



If statements



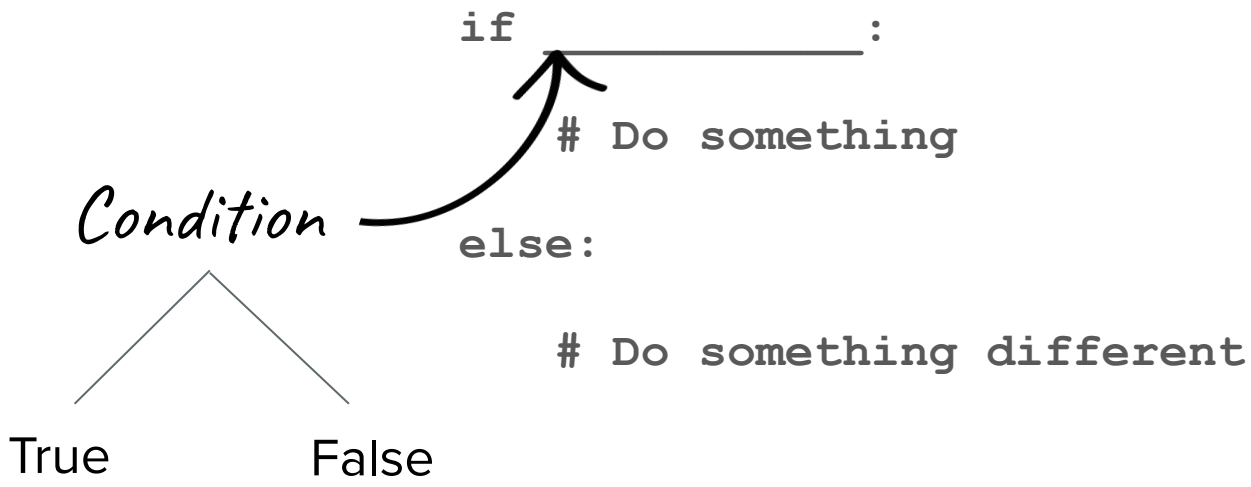
If statements



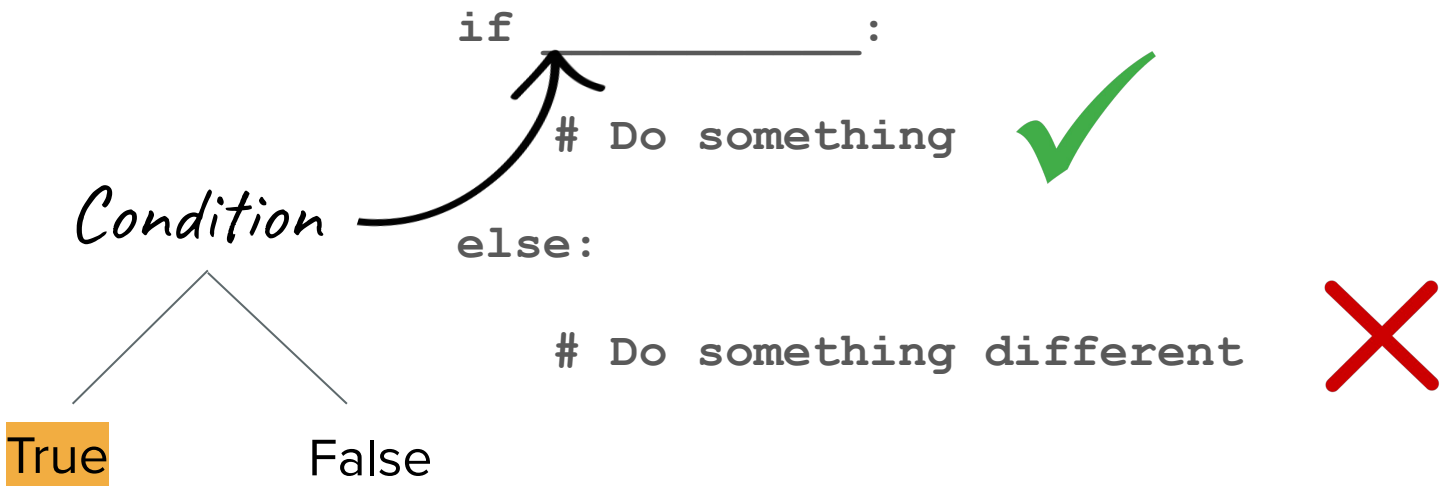
If-else statements

```
if _____________:

    # Do something

else:

    # Do something different
```

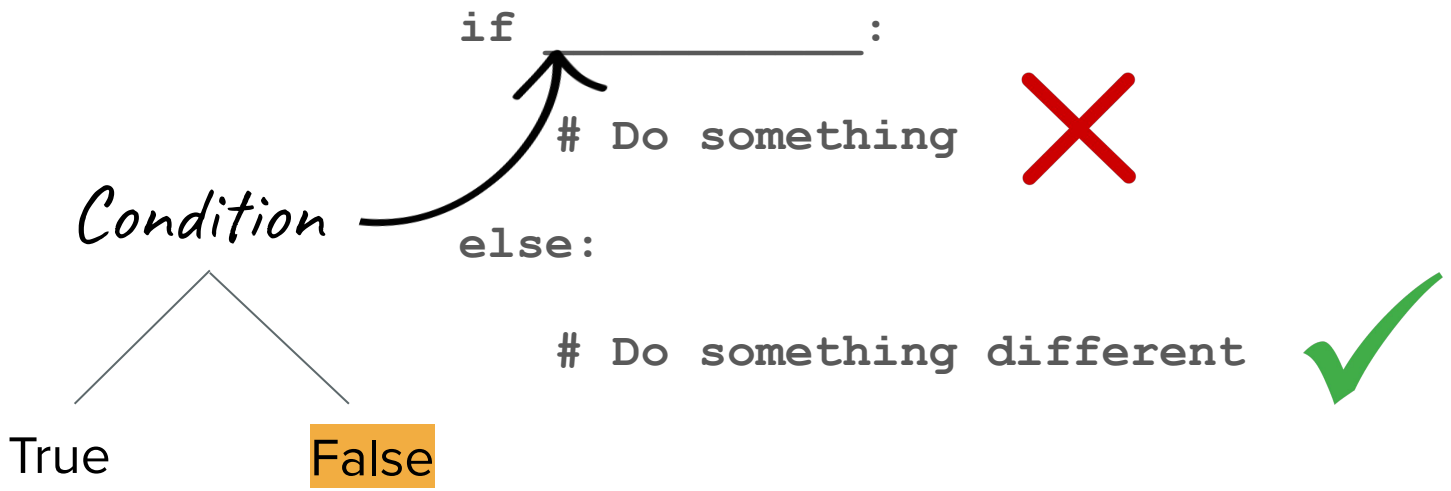
If-else statements



If-else statements

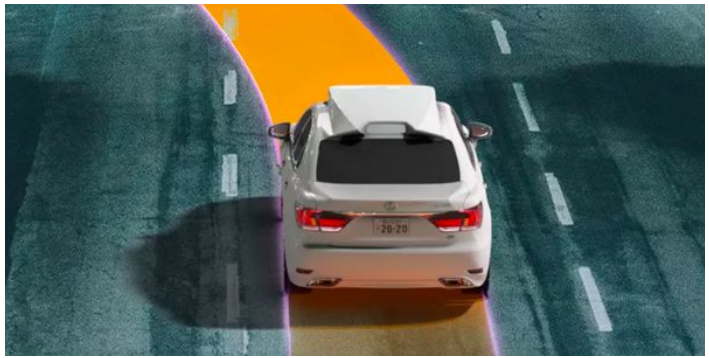


If-else statements

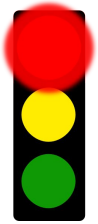


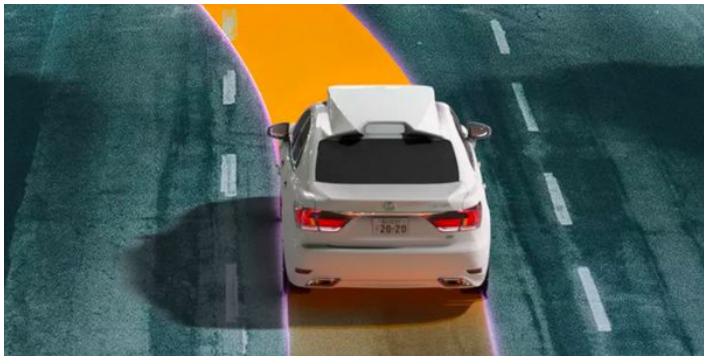
If-else statements

```
if _____________:

    # Do something

else:

    # Do something different
```



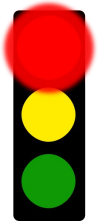
If-else statements



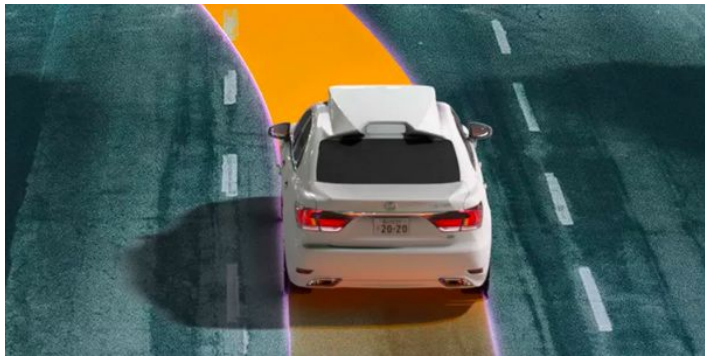


```
if light is red:
    Stop
else:
    Go
```

If-else statements



```
if light is red:
    Stop
else:
    Go
```



What about yellow lights?

Conditionals (if-elif-else)

If-else statements

```
if _____________:

    # Do something

else:

    # Do something different
```

If-elif-else statements

```
if ____________:
    # Do something
elif ____________:
    # Do something different
else:
    # Do another different thing
```

If-elif-else statements

You can have multiple elif blocks.

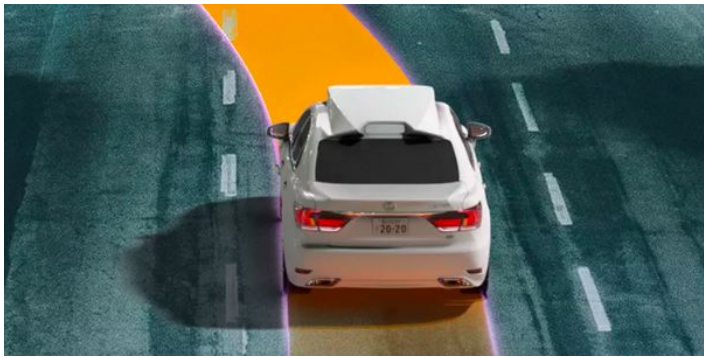
```
if ____________:
    # Do something
elif ____________:
    # Do something different
else:
    # Do another different thing
```

If-elif-else statements

```
if _____________:
    # Do something
elif _____________:
    # Do something different
else:
    # Do another different thing
```

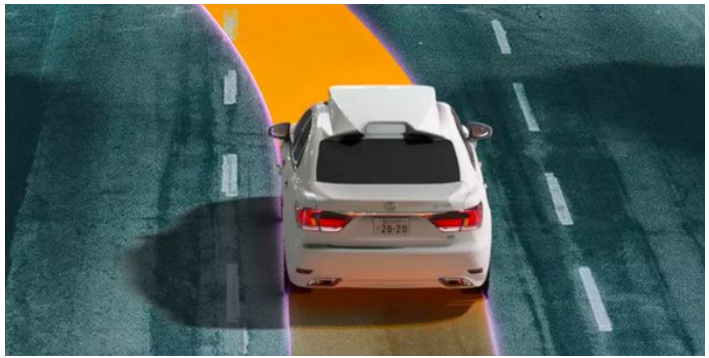
The else is optional!

If-elif-else statements



```
if _____________:
    # Do something
elif _____________:
    # Do something different
else:
    # Do another different thing
```

If-elif-else statements



```
if light is red:
    Stop
elif light is yellow:
    Slow down
else:
    Go
```

If-elif-else statements

- Can have more than one elif block
  - But too many can get messy

- Else is optional
  - You can think of this like a default option

- Put the conditions in priority order!
  - Ask: Which condition do I want to check first?

If-elif-else statements

- Can have more than one elif block
  - But too many can get messy

- Else is optional
  - You can think of this like a default option

- Put the conditions in priority order!
  - Ask: Which condition do I want to check first?

Why not just use multiple if statements?

What's the difference?

```
if light is red:
    Stop
elif light is yellow:
    Slow down
else:
    Go
```

```
if light is red:
    Stop
if light is yellow:
    Slow down
if light is green:
    Go
```

What's the difference?

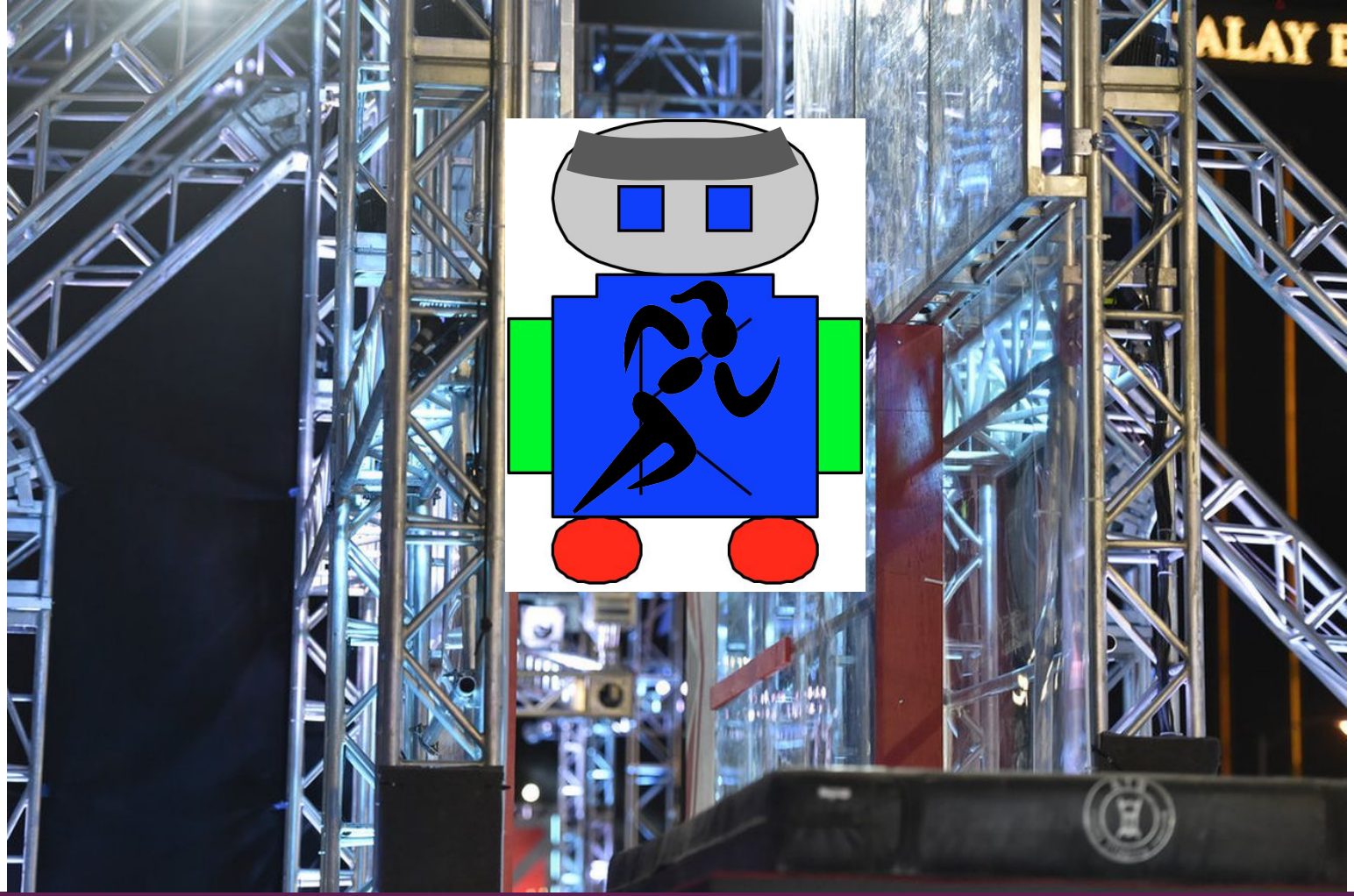
```
if light is red:
    Stop
elif light is yellow:
    Slow down
else:
    Go
```

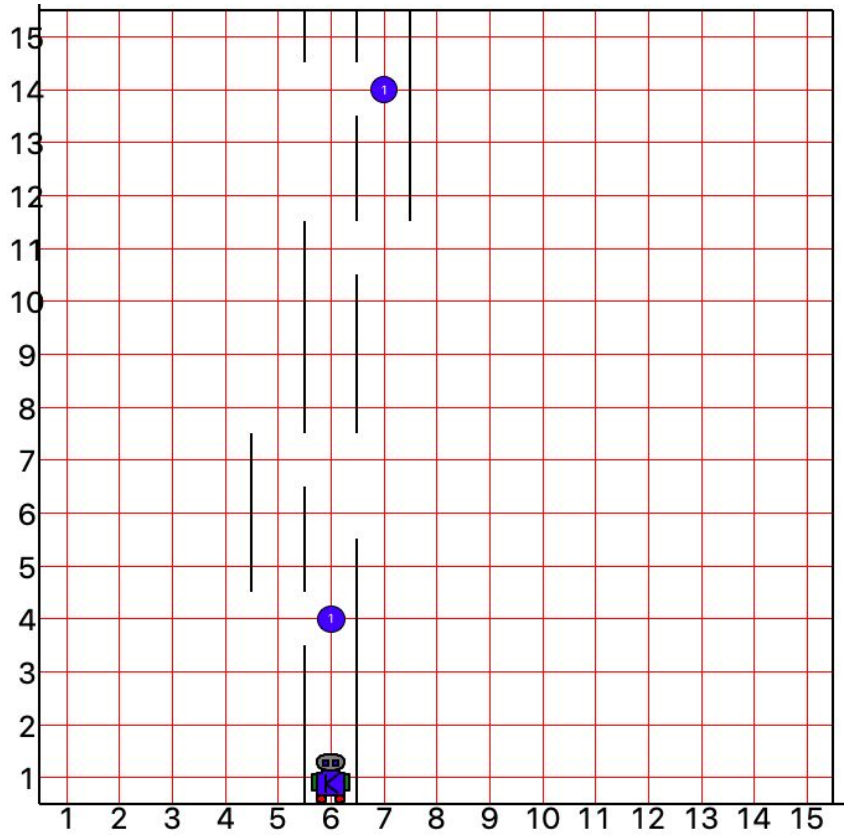
```
if light is red:
    Stop
if light is yellow:
    Slow down
if light is green:
    Go
```

If the light is red, we shouldn't need to check anything else!

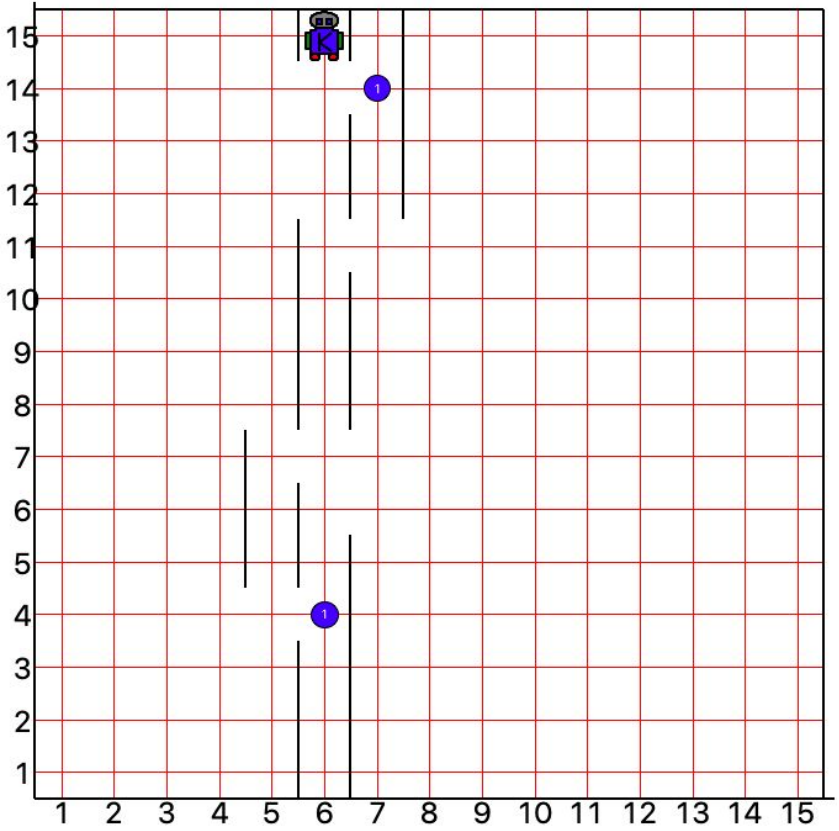
Let’s put it all together!

NinjaKarel

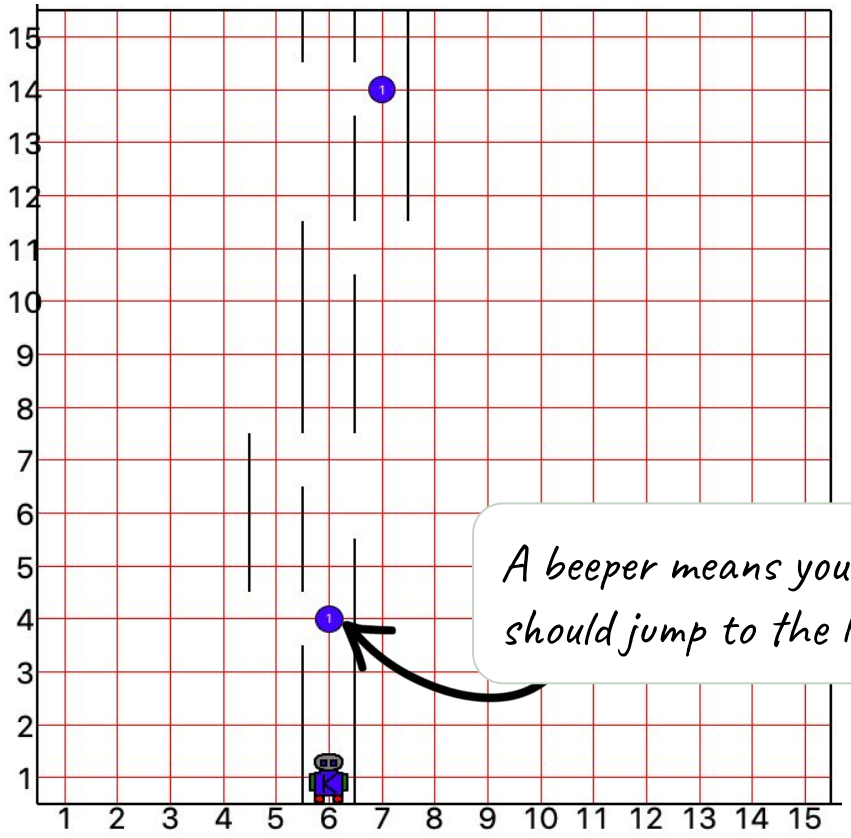



15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

begin

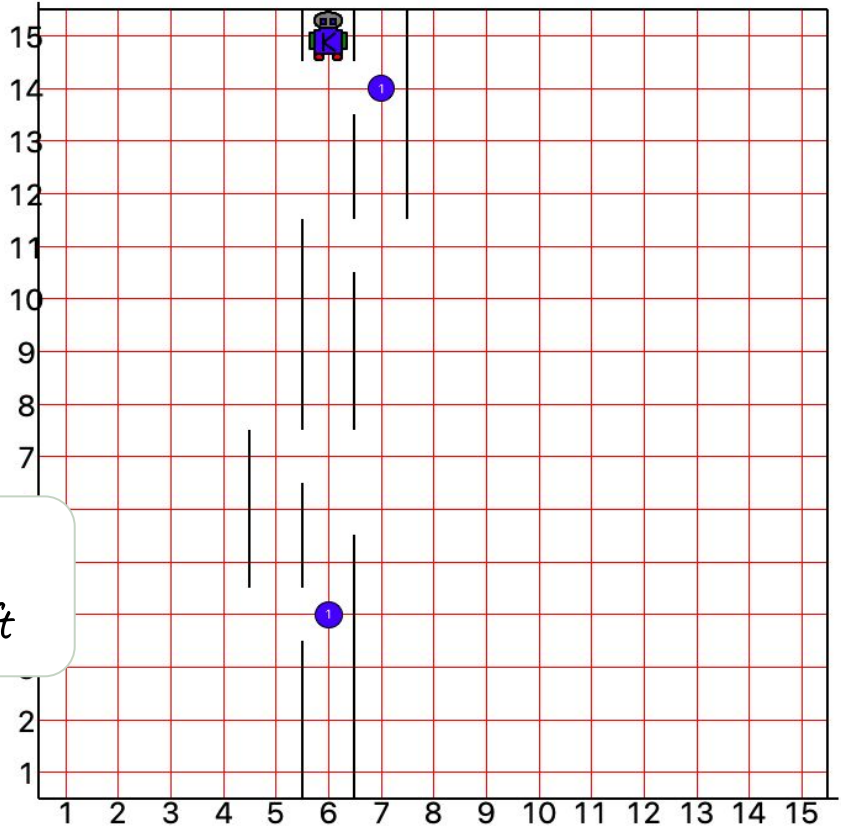

15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

end

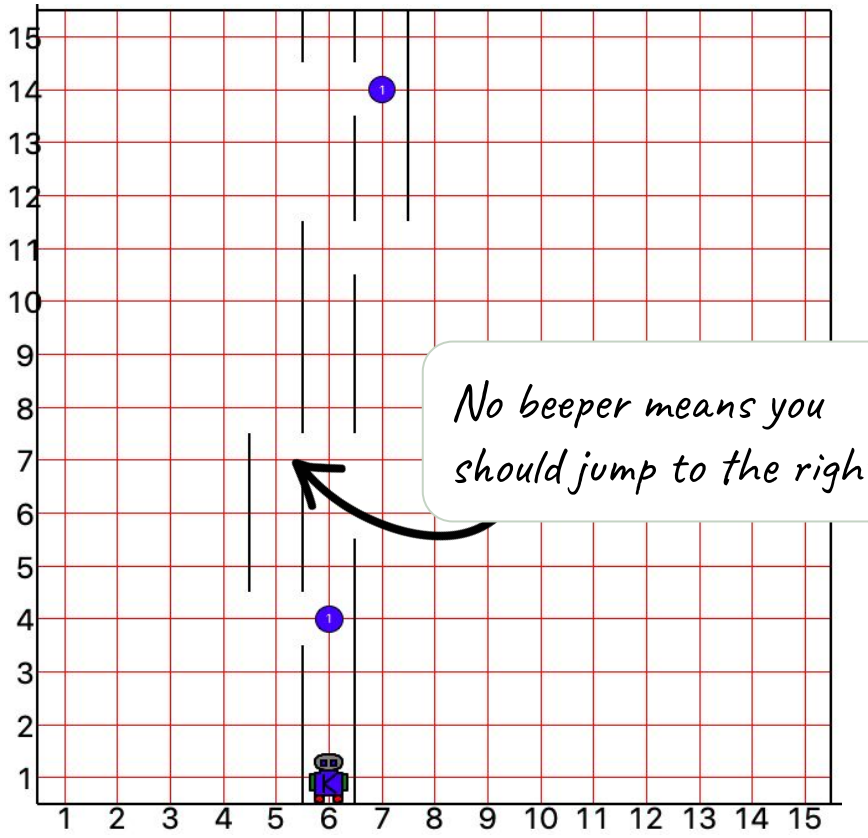


A beeper means you should jump to the left

begin

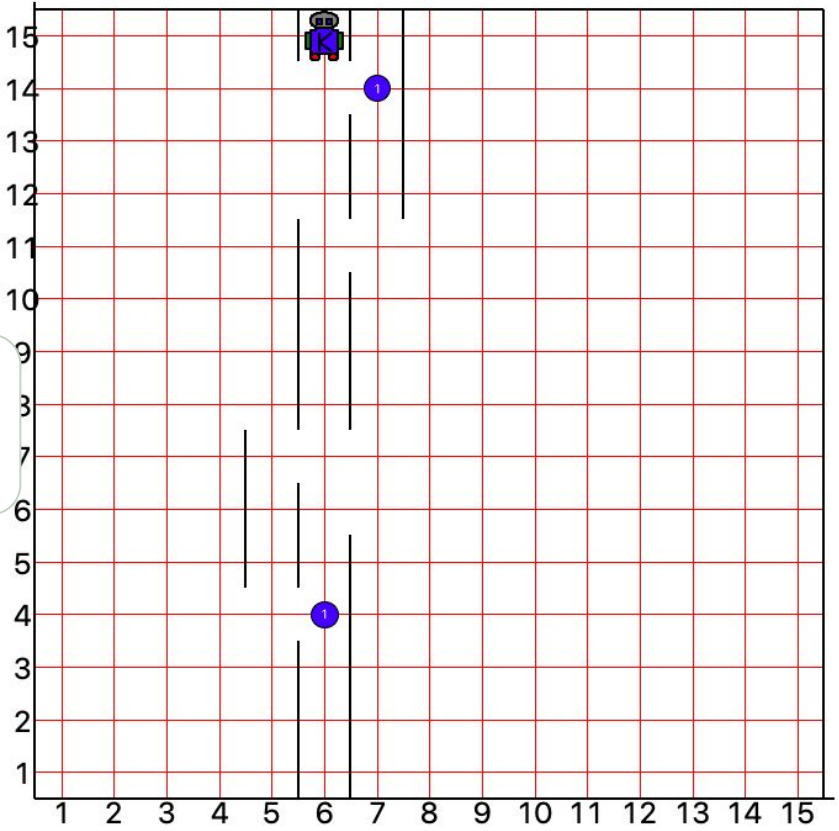


end



No beeper means you should jump to the right

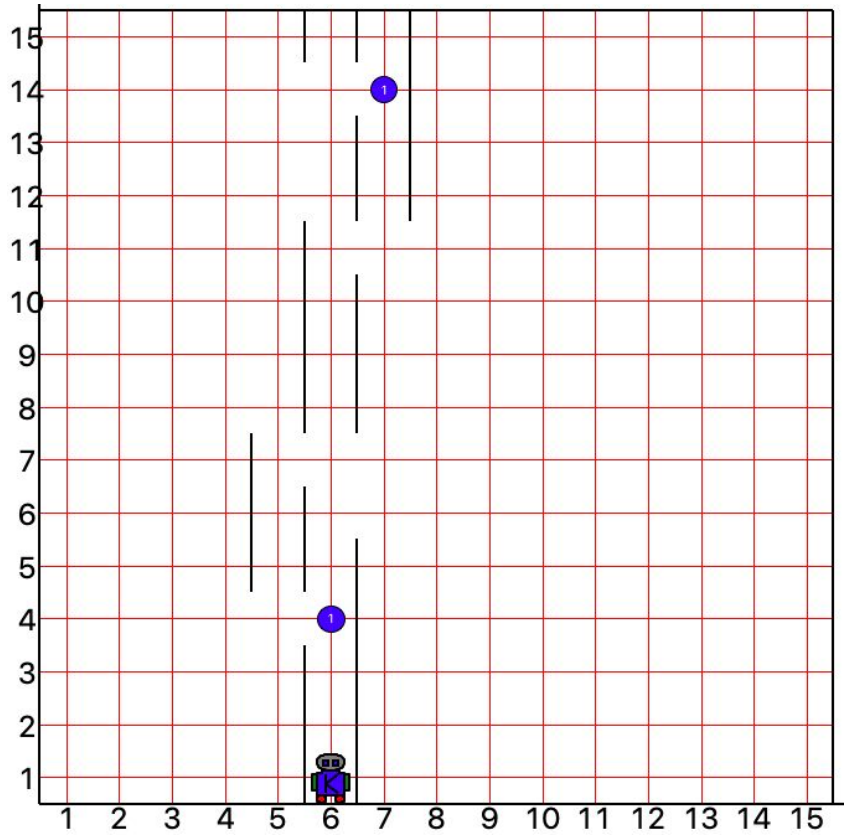
begin



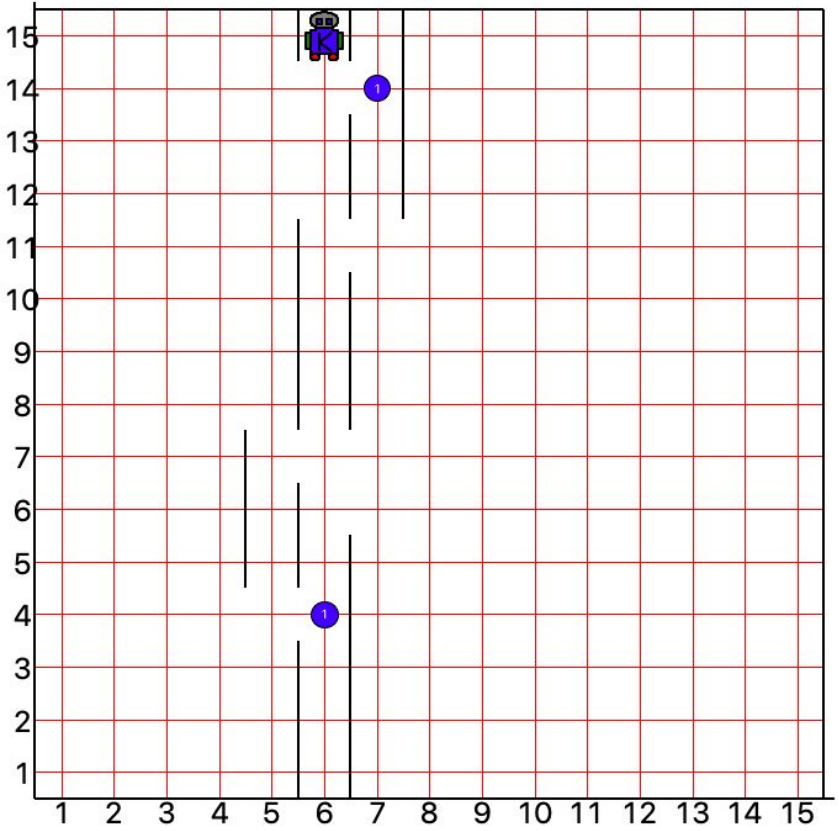
end

NinjaKarel

[demo]


15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

begin

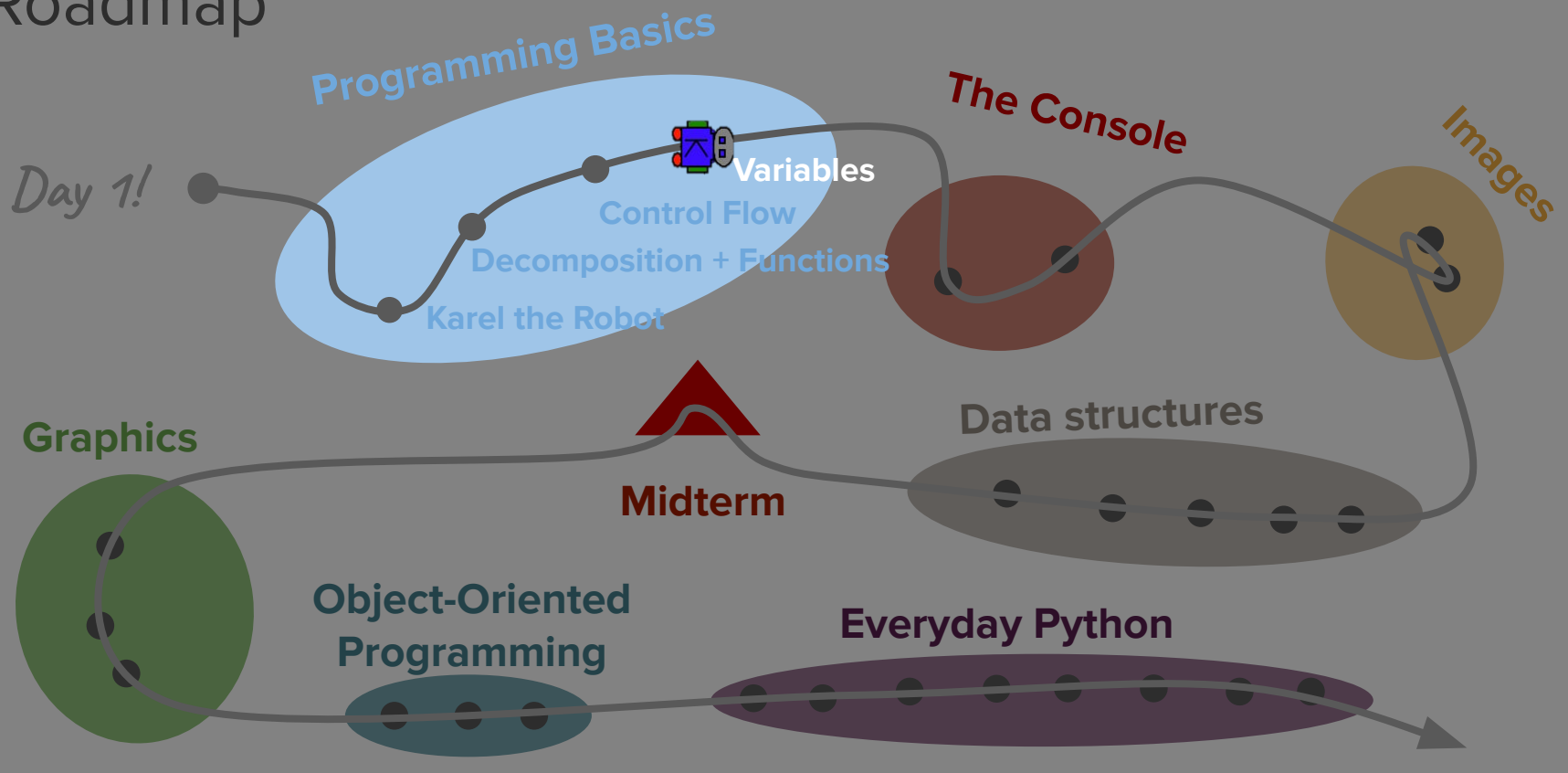

15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

end

What’s next?

You have everything you need for Assignment 1!

Roadmap


Programming Basics
Day 1!
Variables
Control Flow
Decomposition + Functions
Karel the Robot
The Console
Images
Graphics
Midterm
Data structures
Object-Oriented Programming
Everyday Python
Life after CS106AP!