# Algorithms and Data Structures for Data Science

## Formal Logic

CS 277
Brad Solomon

August 25, 2021

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

mp_racing delayed a week

**Key Concepts:**

- Python Control Structures
- Data Parsing
- Logic

12* lab assignments (15 points each)

5 machine problem sets (60 points each)

# Reminder: Labs are online

https://courses.grainger.illinois.edu/cs277/fa2021/info/labs/

Two lab sections: 10 AM and 11 AM

# Joining class Github

https://edu.cs.illinois.edu/create-ghe-repo/cs277-fa21/

**Your CS 277 repository is ready!**

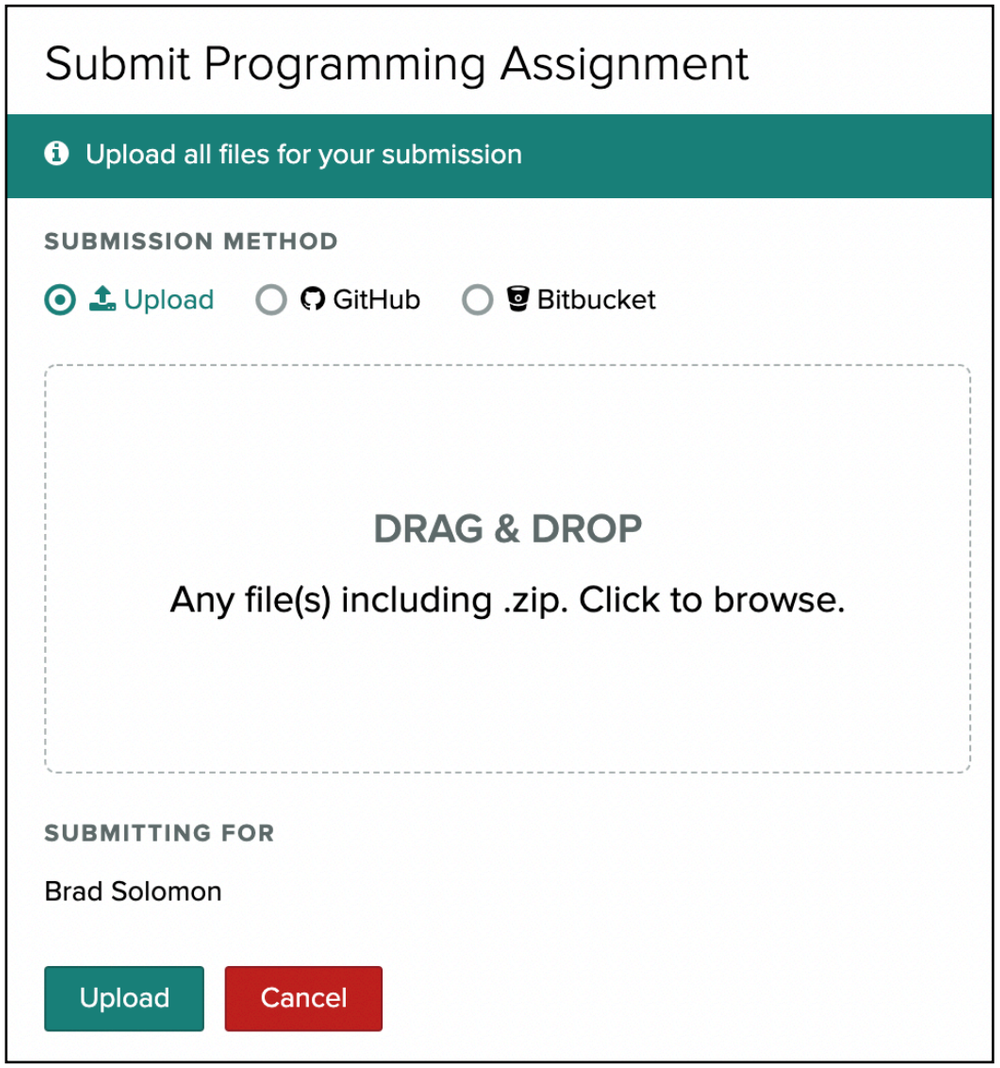
You will submit all course assignments through this repository:

- `https://github-dev.cs.illinois.edu/cs277-fa21/`[redacted]
- Follow your course-specific instructions on what to do next (you will usually not visit your git URL directly).

# Submitting via Gradescope

Must upload directly

Only submit files listed on each assignment



# Last time:

## Fundamental of Python programming

### Selection Control Structures

controlStructs.py

```
num = 20

if num in [0,1,2,3,4]:
    print("Top 5!”)

elif num > 10:
    print("num too large!

elif num > 15:
    print("will this ever

else:
    print(num)
```

### Functions

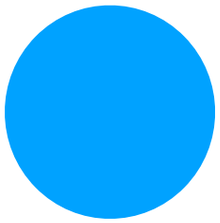
**Functions organize code with clear I/O**

```
def addTogether(x, y, z =

    if z == None:
        return x + y
    else:
        return x + y + z

if __name__ == '__main__':

    print(addTogether(2,3)
    print(addTogether(2,3,
```

### Object-Oriented Programming

An **object** is a conceptual grouping of variables and functions that make use of those variables

Variables:



Functions:

# Object-Oriented Programming

*An **object** is a conceptual grouping of variables and functions that make use of those variables*

Variables:

Functions:

# Object-Oriented Programming

*'self' generically represents the **instance** of the class*

```
from random import randint

class gacha:
    def __init__(self):
        pass


    # Input: Nothing!
    # Output: The random prize that was won
    # Should also modify cash reserve and prizeList
    def pullPrize(self):
        pass



if __name__ == '__main__':
    pass
```

gacha_class.py

# Object-Oriented Programming

Variables:

Functions:

# Object-Oriented Programming

## *Classes can redefine Python built-in functions*

```
class prize:
    def __init__(self, uid, name, rarity):
        self.rarity = rarity
        self.name = name
        self.uid = uid

    # Representation of object
    def __repr__(self):
        return "({}, {}, {})".format(self.uid, self.name,
self.rarity)

    # String Representation of object
    def __str__(self):
        return "({}, {}, {})".format(self.uid, self.name,
self.rarity)
```

gacha.py

# Storing a CSV

```
ID, Area, Beds, Baths, Zip, Year, Price

 1, 2222, 3, 3.5, 32312, 1981, 250000
 2, 1628, 3, 2,   32308, 2009, 185000
 3, 3824, 5, 4,   32312, 1954, 399000
```

# Accessor Functions

*Objects can be used purely to store information*

```
class housingData:
    def __init__(self):
        pass
```

housing.py

# Learning Objectives

Formalize logic expressions and operators

Construct and solve conditional expressions

Motivate and explore quantificational logic

# Why formal logic?

Fundamentally useful to CS:

- Coding control structures
- Set abstract data type
- Proofs and theory

Many applications in the real world

# Radical Racing Robots

column

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

row

Robot starts at (2,1)

# Radical Racing Robots

column

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

row

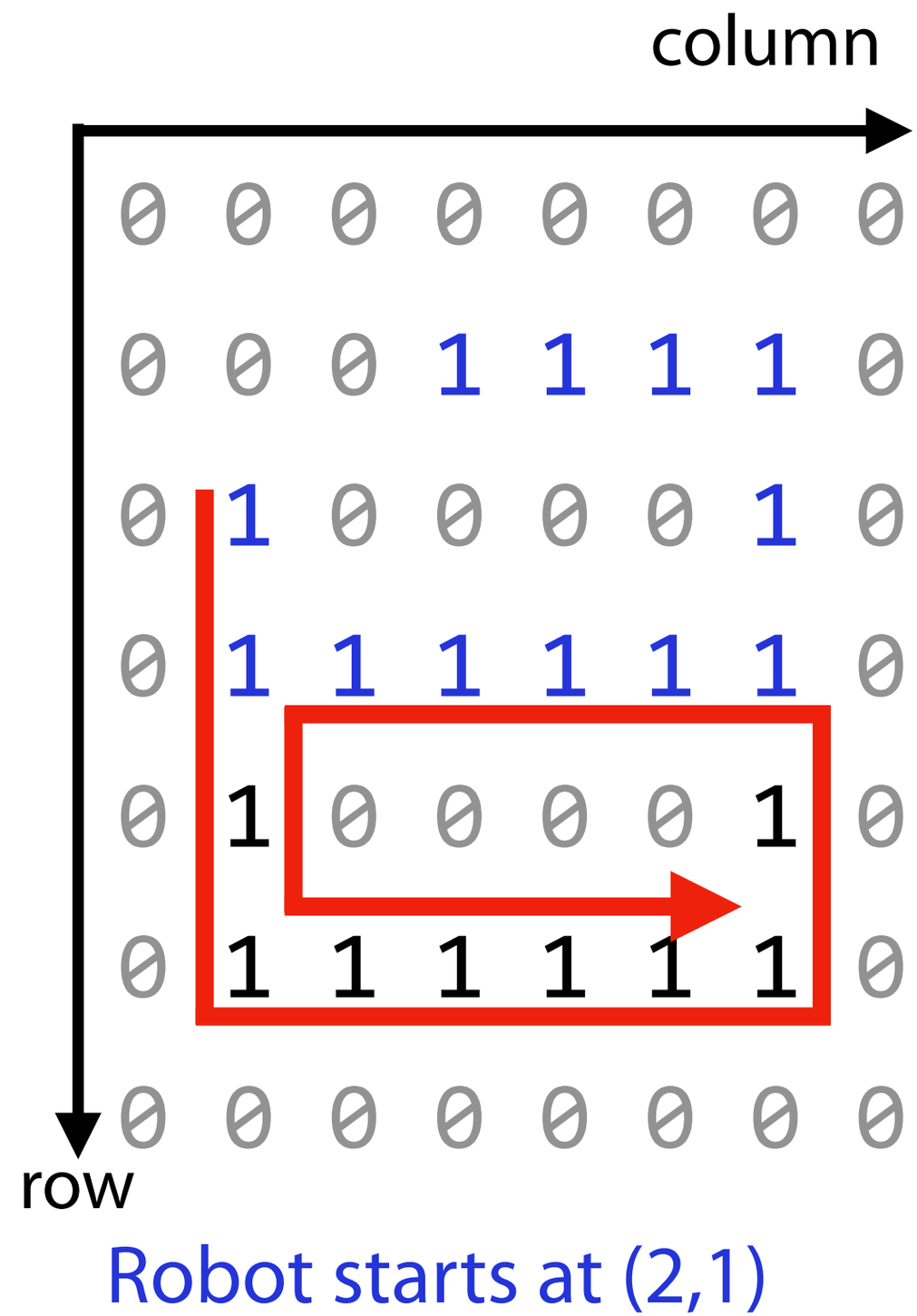
Robot starts at (2,1)

# Radical Racing Robots

1) Practice writing conditionals and loops

2) Process a multi-file data source

3) Use formal logic to code a virtual robot



# Propositional Logic

*A declarative sentence which is either True or False*

*(but not both)*

This sentence is a proposition. States a fact ✓ True! ✓

Computer science is boring. States a fact ✓ False! ✓

This statement is false. States a fact ✓ True! False! True! False! …

# Propositional Logic

Which of the following are propositions?

Data science is the best science.

The cake is a lie.

$1 + 1 = 3$

Beware the Jabberwock

Python is a language and an animal

# Propositional Logic

We can make more complex propositions with ***operators***

Python is a language

AND

Python is an animal

$1 + 1 = 3$

OR

Pineapple on pizza is delicious

An apple a day does NOT keep the doctor away

# Logic Operators *(Not)*

The **negation** of p is denoted as $\neg p$

| $p$ | $\neg p$ |
|---|---|
| T | |
| F | |

# Logic Operators *(And)*

The **conjunction** of p and q is denoted as $p \wedge q$

| $p$ | $q$ | $p \wedge q$ |
|---|---|---|
| T | T | |
| T | F | |
| F | T | |
| F | F | |

# Logic Operators *(Or)*

The **disjunction** of p and q is denoted as $p \vee q$

| $p$ | $q$ | $p \vee q$ |
|---|---|---|
| T | T | |
| T | F | |
| F | T | |
| F | F | |

# Logic Operators *(xor)*

The **exclusive or** of p and q is denoted as $p \oplus q$

| $p$ | $q$ | $p \oplus q$ |
|---|---|---|
| T | T | |
| T | F | |
| F | T | |
| F | F | |

# Logic Operators

*Logic operators modify or join propositions together*

| English | Logic | Python |
|---|---|---|
| **Not** $p$ (Negation) | $\neg p$ | |
| $p$ **and** $q$ (Conjunction) | $p \wedge q$ | |
| $p$ **or** $q$ (Disjunction) | $p \vee q$ | |
| $p$ **xor** $q$ (Exclusive Or) | $p \oplus q$ | |

# Logical Equivalencies

**Commutative**

$$(p \wedge q) \equiv (q \wedge p)$$

$$(p \vee q) \equiv (q \vee p)$$

# Logical Equivalencies

## Distributive

$$p \wedge \left(q \vee r\right) \equiv$$

$$p \vee \left(q \wedge r\right) \equiv$$

# De Morgans Laws

How do we negate a complex proposition?

$$\neg\left(p \wedge q\right) \equiv$$

# De Morgans Laws

How do we negate a complex proposition?

$$\neg\left(p \vee q\right) \equiv$$

# Propositional Logic

Complex propositions can sometimes get confusing

$$a = True \quad b = False \quad c = True \quad d = True$$

$$a \wedge b \vee c \wedge d$$

# Order of Logic Operators

Highest Precedence

$\neg p$

$p \wedge q$

$p \vee q$

Lowest Precedence

# Logic Exercise 1

$p$: It is raining outside.

$q$: It is snowing outside.

$r$: I am wearing a jacket.

$s$: I am cold.

$$\neg p \wedge \neg s$$

$$r \wedge (p \vee q)$$

$$(p \oplus q) \wedge (r \vee s)$$

# Logic Exercise 2

**Write a logic statement:** Either apples and oranges grow on trees or money grows on trees, but not both.

$a$: Apples grow on trees

$b$: Oranges grow on trees

$c$: Money grow on trees

# Logic Exercise 3

**“Does everyone want coffee?”**

| Nancy | Carl | Tiffani |
|---|---|---|
|  |  |  |
| ‘I don’t know’ | ‘I don’t know’ | ‘No.’ |

Who wants coffee?

What is the logic expression?

# Conditional Statements

A conditional statement is a compound proposition that means "if p then q" and is denoted as $p \Longrightarrow q$

| $p$ | $q$ | $p \Longrightarrow q$ |
|---|---|---|
| T | T | |
| T | F | |
| F | T | |
| F | F | |

# Logical Equivalent to Conditional

Given $p$ and $q$, can we rewrite an equivalent compound proposition to the conditional $p \implies q$?

| $p$ | $q$ | |
|---|---|---|
| T | T | T |
| T | F | F |
| F | T | T |
| F | F | T |

# De Morgans Laws for Conditionals

How do we negate a conditional proposition?

$$\neg\left(p \Longrightarrow q\right) \equiv$$

# Converse of a conditional

The **converse** of a conditional statement is formed by swapping the position of its variables.

What is the converse of $p \implies q$?

| $p$ | $q$ | |
|---|---|---|
| T | T | |
| T | F | |
| F | T | |
| F | F | |

# Contrapositive

The **contrapositive** of a conditional statement is formed by swapping the position of its variables and negating both of them.

| $p$ | $q$ | $p \Longrightarrow q$ | |
|---|---|---|---|
| T | T | T | |
| T | F | F | |
| F | T | T | |
| F | F | T | |

# Logic Exercise 4

$p$: It is freezing outside.

$q$: My car will start

"If it's freezing outside, my car won't start"

Logic:

Converse:

Contrapositive:

# Biconditional Statements

A biconditional statement is a compound proposition that means "p if and only if q" and is denoted as $p \iff q$

| $p$ | $q$ | $p \iff q$ |
|---|---|---|
| T | T | |
| T | F | |
| F | T | |
| F | F | |

# Logical Equivalent to Biconditional

Given $p$ and $q$, can we rewrite an equivalent compound proposition to the biconditional?

$$p \Longleftrightarrow q \equiv$$

# Logic Exercise 5

**Write a logic statement:** You will get an 'A' in this class if and only if you complete all assignments, ask questions in class, and either take all exams or get all extra credit points.

# A twist on the two-guard riddle

*One guard tells the truth, the other guard lies.*

**Guard 1:** “The exit is behind this door and death is behind the other door”

**Guard 2:** “The exit is behind a door and death is behind the other door”

# A twist on the two-guard riddle

*One guard tells the truth, the other guard lies.*

**Guard 1:** “The exit is behind this door and death is behind the other door”

**Guard 2:** “The exit is behind a door and death is behind the other door”

$$\Longrightarrow$$

# A twist on the two-guard riddle

*One guard tells the truth, the other guard lies.*

**Guard 1:** “The exit is behind this door and death is behind the other door”

**Guard 2:** “The exit is behind a door and death is behind the other door”

$$\Longrightarrow$$

# Propositions vs Predicates

*A **predicate** is a sentence with variables which can be turned into a proposition when assigned values*

```
x = 5
y = "Bees"
z = [1, 2, 5]

# Propositions
print(x == 3)

print("b" in "Bees")

print(len(z)> 1)
```

prop_vs_pred.py

# Predicate Logic

*A **predicate** is a sentence with variables which can be turned into a proposition when assigned values*

$P(x)$: "$x$ goes to UIUC"

$Q(x)$: $x^2 + 5 \geq 10$

# Predicate Logic

*A **predicate** is a sentence with variables which can be turned into a proposition when assigned values*

$P(x)$: "$x$ is odd."

# Domain of Discourse $\left(P(x), x \in \{D\}\right)$

*The set of values we allow a variable to take*

**1) Constrain a predicate to a set of values**

$P(x) = \text{“}x < 5\text{”}$ (True for an infinite number of values)

# Domain of Discourse $\left(P(x), x \in \{D\}\right)$

*The set of values we allow a variable to take*

**1) Constrain a predicate to a set of values**

$P(x) = \text{“}x < 5\text{”}$ (True for an infinite number of values)

$P(x) = \text{“}x < 5\text{”}, x \in \mathbb{N}$

# Domain of Discourse $\left(P(x), x \in \{D\}\right)$

*The set of values we allow a variable to take*

**1) Constrain a predicate to a set of values**

$P(x) = \text{“}x < 5\text{”}$ (True for an infinite number of values)

$P(x) = \text{“}x < 5\text{”}, x \in \mathbb{N}$ (True for {0,1,2,3,4}, a finite set!)

‘x is in the domain of natural numbers (0, 1, 2, …)’

# Domain of Discourse $\left(P(x), x \in \{D\}\right)$

*The set of values we allow a variable to take*

**1) Constrain a predicate to a set of values**

**2) Prevent nonsense statements** (Neither True nor False)

$P(x) =$ "10 / $x$ > 1"

$x =$ Any value that isnt a number

E.g. '10 / Zebra > 1'

$x = 0$

E.g. '10 / 0 > 1'

# Domain of Discourse $\left(P(x), x \in \{D\}\right)$

*The set of values we allow a variable to take*

**1) Constrain a predicate to a set of values**

**2) Prevent nonsense statements** (Neither True nor False)

$P(x) =$ "10 / $x$ > 1", $x \in \mathbb{R}_{\neq 0}$ (-e, $\pi$ , 0.5, 12/5, …)

$x =$ Any value that isnt a number

E.g. '10 / Zebra > 1'

$x = 0$

E.g. '10 / 0 > 1'

# Predicate Programming Logic

myList = `[3, 6, 7, 2, 1]`

*"Return all values which are even in myList"*

**Domain:**

**Variables:**

**Predicates:**

# Predicate Programming Logic

| | | | |
|---|---|---|---|
| 4 | 3 | 2 | 3 |
| 1 | 1 | 1 | 1 |
| 5 | 2 | 1 | 1 |
| 2 | 0 | 8 | 1 |

*"Return the first row in the matrix with at least two 1s in it."*

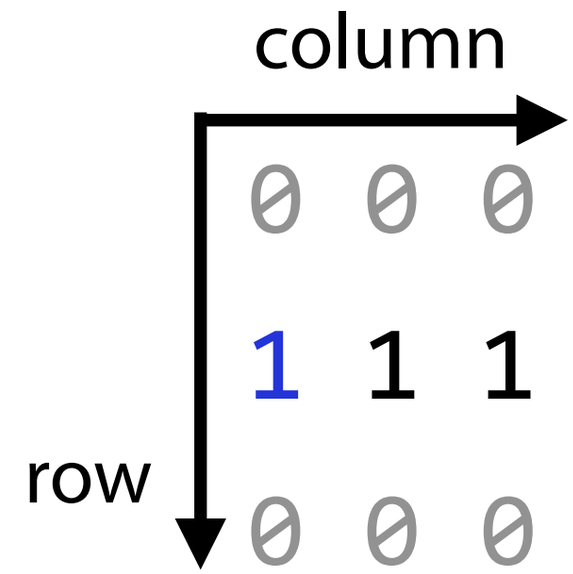
**Domain:**

**Variables:**

**Predicates:**

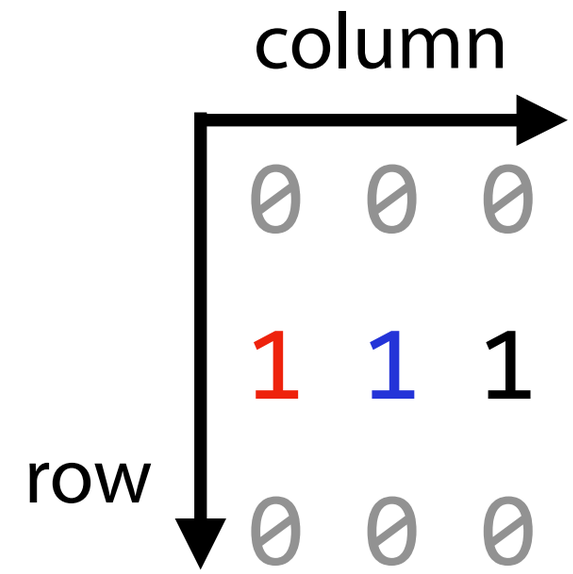
# Radical Racing Robots

*"RacingBot should move one square in a cardinal direction away from its current position."*



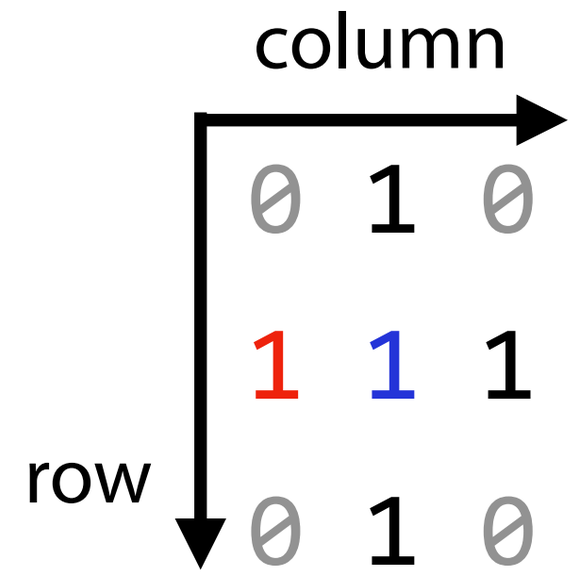
# Radical Racing Robots

*"RacingBot should move one square in a cardinal direction away from its current position* ***but cannot move backwards****."*



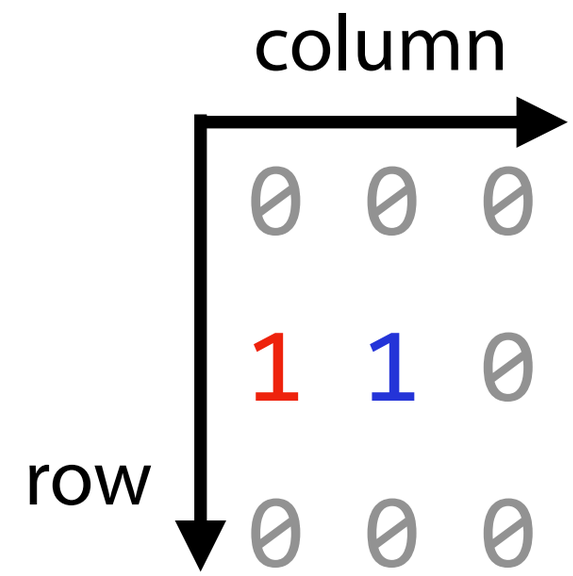
# Radical Racing Robots

*"If multiple moves are available, move in a clockwise order but don't move backwards from our current position."*



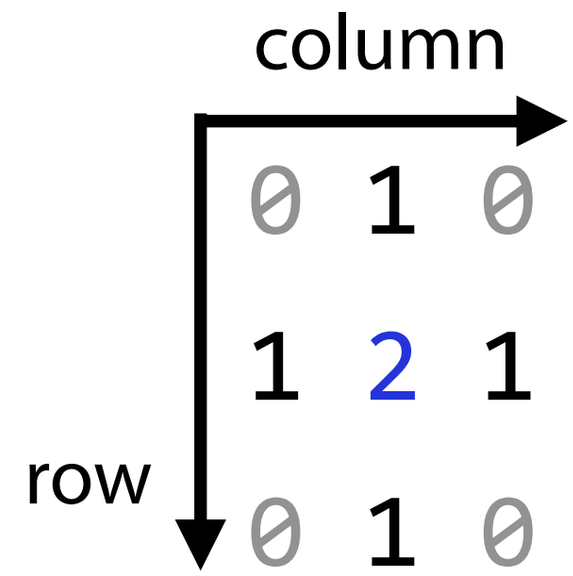
# Radical Racing Robots

*"If there are no valid moves, racingBot should stay in place."*



# Radical Racing Robots

*"If racingBot's current position has a value greater than 1, it's movement is based on the value of it's cargo."*




"If there exists three unique cargo objects $x, y, z$ such that $x = y + z$, racingBot should move north"