Impact Lab 2025: Programming Fundamentals
Lecture 3: `random()` and Objects

Summer 2025

School of Computing
and Data Science

Wentworth Institute of
Technology

Topics for Today

- `random()`
- Object Oriented Programming

`random()` Function

```
function setup(){
  print(random(-50,50));
}
```

`random()` creates a random floating-point number.

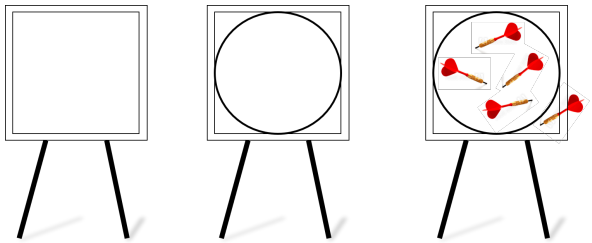
It can take 0, 1, or 2 arguments (or an array).

Let's look at the documentation

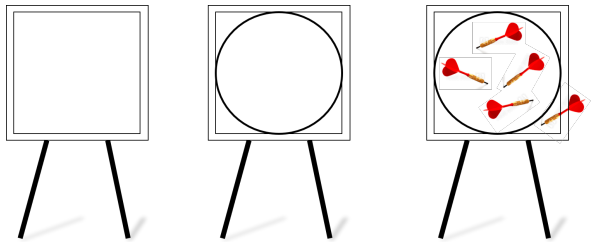
Exercise: Discovering PI

- PI is the ratio of a circle's circumference to its diameter.
- Its approximate value is 3.14159265...

There are many ways to estimate its value, but today, we're going to use the "Dart Board" approach.



Discovering PI



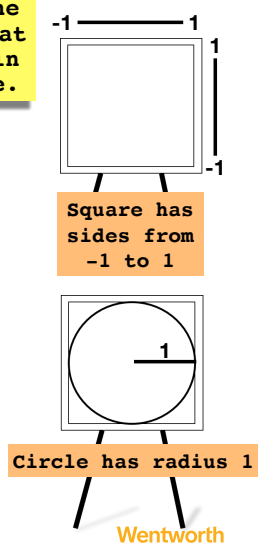
The Idea is simple:

- Start with a square.
- Add a circle inside (exactly touching the edges).
- Throw darts at the square randomly.
- Count how many land in the circle vs how many total darts you throw.
- The result (hits in the circle/total darts) will be related to PI.

Throw a Dart

```
let x=random(-1,1);
let y=random(-1,1);
```

"Throw" the dart so that it lands in the square.

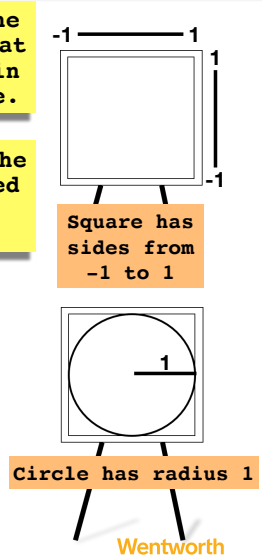


Throw a Dart

```
let x=random(-1,1);
let y=random(-1,1);

if(Math.sqrt(x*x+y*y)<=1){
  inCircle+=1;
}
```

"Throw" the dart so that it lands in the square.

Check if the dart landed in the circle



Compare to Circle

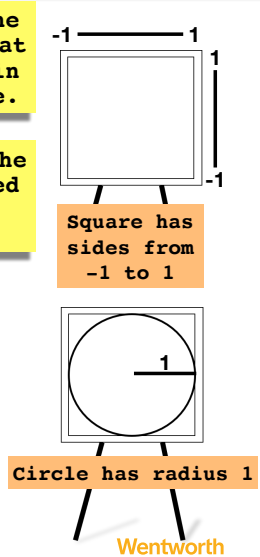
```
let x=random(-1,1);
let y=random(-1,1);

if(Math.sqrt(x*x+y*y)<=1){
  inCircle+=1;
}
```

"Throw" the dart so that it lands in the square.

Check if the dart landed in the circle

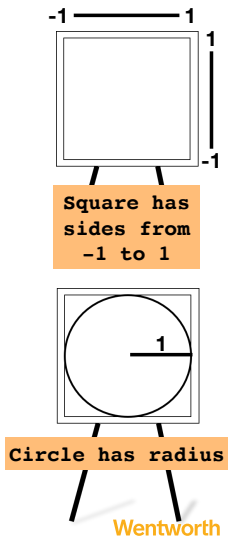
Do this many times (in a loop) and report how many landed in the circle.

Divide by the total number of darts thrown.



Do Many Times

```
let N=1000000;
let inCircle=0;

for(let i=0;i<N;i++)
{
  let x=random(-1,1);
  let y=random(-1,1);

  let inCircle=0;
  if(Math.sqrt(x*x+y*y)<=1){
    inCircle+=1;
  }
}
```



Why Does This Work (And What to Print)

Area of a Circle:

$$A_c = \pi r^2$$

Area of a Square:

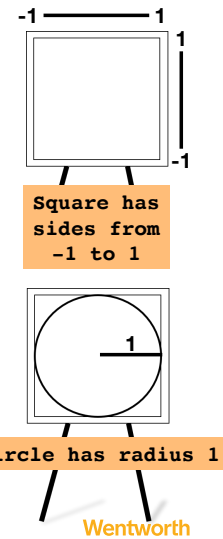
$$A_s = D^2 = 4r^2$$

Ratio:

$$\frac{A_c}{A_s} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4}$$

So, if we throw enough darts, they approximate the area of both the square and circle.

$$\pi = 4\frac{H}{N}$$



Let's Make a Deal

Let's Make a Deal is a game show that started in 1963 and has appeared in various forms up to today.

The final segment of the original show was "The Big Deal" and it's what we're going to be simulating today.

Monte Hall

Our goal today is to gain **insight** about this final game show segment.

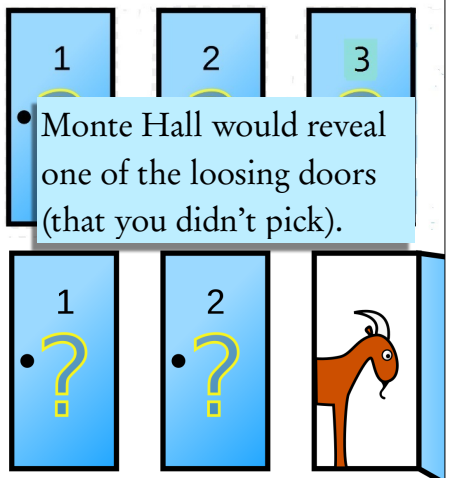
The Big Deal

The final segment of the show always contained a big item, like a car, vacation, etc.

The player would select one of three doors.

Behind two would be a goat (or similar), but behind one was the prize.

He then gives you the chance to change your pick.



The Big Deal

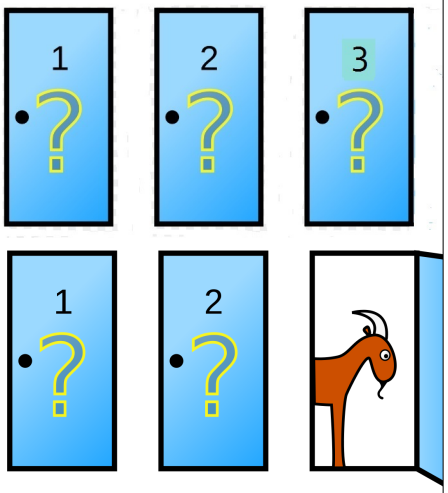
Let's say you pick door 2.

Monte Hall shows you what's behind door 3.

Do you change your pick to door 1?

Is it better to switch or stay?

What is the chance of winning if you stay or if you switch?



Wentworth
Computing & Data Science

The Big Deal

First, let's think about how we would run this game once on our computer:

```
function setup() {

  let winningDoor = int(random(3));
  let playerDoor = int(random(3));

  if(winningDoor == playerDoor) {
    print("Win");
  } else {
    print("Lose");
  }
}
```

Generate two random numbers, one for the winning door and one for the door that the player picks, labeled 0, 1, and 2.

If the player selects the winning door, we print "Win", otherwise we print "Lose"

We'll add in switching in a moment

What do you think the "**`int()`**" is doing?

Wentworth
Computing & Data Science

The Big Deal

But this is only one run of the game, how can we "see" that we have a 1/3 chance to win and a 2/3 chance to lose?

Monte Carlo Simulation

Run the game many times, counting the wins and loses, and compute the probabilities at the end.

```
function setup() {

  let winningDoor = int(random(3));
  let playerDoor = int(random(3));

  if(winningDoor == playerDoor) {
    print("Win");
  } else {
    print("Lose");
  }
}
```

Wentworth
Computing & Data Science

The Big Deal

But this is only one run of the game, how can we "see" that we have a 1/3 chance to win and a 2/3 chance to lose?

Monte Carlo Simulation

Run the game many times, counting the wins and loses, and compute the probabilities at the end.

```
function setup() {
  let runs=100;
  let winCount=0;
  let loseCount=0;

  for(let i=0;i<runs;i++){
    let winningDoor = int(random(3));
    let playerDoor = int(random(3));

    if(winningDoor == playerDoor) {
      print("Win");
      winCount++;
    } else {
      print("Lose");
      loseCount++;
    }
  }
}
```

Wentworth
Computing & Data Science

The Big Deal

```
function setup() {
  let runs=100;
  let winCount=0;
  let loseCount=0;

  for(let i=0;i<runs;i++){
    let winningDoor = int(random(3));
    let playerDoor = int(random(3));

    if(winningDoor == playerDoor) {
      print("Win");
      winCount++;
    } else {
      print("Lose");
      loseCount++;
    }
  }
  let winProb=winCount/runs;
  let loseProb=loseCount/runs;
  print("Win Prob: "+winProb);
  print("Lose Prob: "+loseProb);
}
```

What's the Probability?

The number of wins or loses divided by the total number of runs that we did.

Revealing

Let's change up the code slightly, we are really only interested in how often we win if we switch vs if we stay.

```
let runs=100;
let winStay=0;
let winSwitch=0;

for(let i=0;i<runs;i++){
  let winningDoor = int(random(3));
  let playerDoor = int(random(3));

  //reveal door
  let revealDoor = 0
  if(revealDoor==playerDoor || revealDoor==winningDoor) {
    revealDoor = 1;
  }
  if(revealDoor==playerDoor || revealDoor==winningDoor) {
    revealDoor = 2;
  };
}
```

There are many ways to do this, we'll take a simple approach:

Pick a door, see if it works, then try another door.

Switching

For switching, we'll do the same thing, except that we can only switch to a door that hasn't been revealed and isn't currently chosen.

```
for(let i=0;i<runs;i++){
  let winningDoor = int(random(3);
  let playerDoor = int(random(3);

  //reveal door
  let revealDoor = 0
  if(revealDoor==playerDoor || revealDoor==
    revealDoor = 1;
  }
  if(revealDoor==playerDoor || revealDoor==winningDoor) {
    revealDoor = 2;
  }

  //Switch door
  let switchDoor = 0
  if(switchDoor==playerDoor || switchDoor==revealDoor) {
    switchDoor = 1;
  }
  if(switchDoor==playerDoor || switchDoor==revealDoor) {
    switchDoor = 2;
  }
}
```


Lastly

Now we just need to keep track of the times when we would have won if we stayed with our original choice of if it was better to switch.

```
for(let i=0;i<runs;i++){
  ...

  if(winningDoor == playerDoor){
    winStay++;
  }
  if(winningDoor == switchDoor){
    winSwitch++;
  }
}
print("Win Stay: "+winStay/runs);
print("Win Switch: "+winSwitch/runs);
```

```
Win Stay: 0.33
Win Switch: 0.67
```

Our new insight:

We win twice as often if we switch.

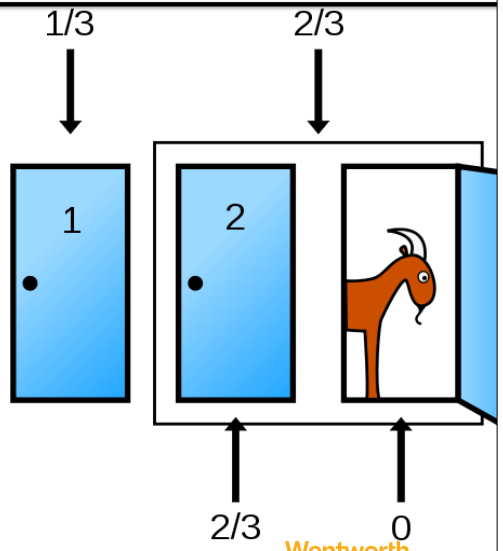
Always Switch!

Why Does it Work This Way?

If we pick door 1, we have 1/3 chance to win, with 2/3 chance that the winning door is 2 or 3.

When door 3 is revealed, we know it was not a winner, but the probability of those two doors stay the same.

So, door 2 has 2/3 chance to be a winner.



Topics for Today

- `random()`
- **Object Oriented Programming**

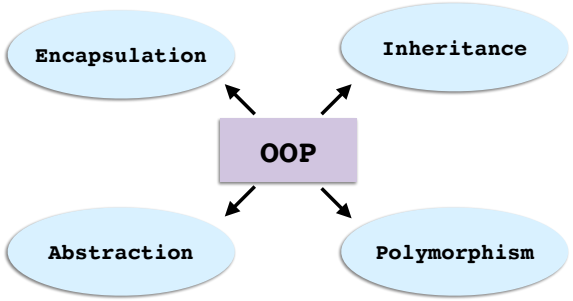
Object Oriented Programming (OOP)

Paradigm based on data types:

- Identify **things** that are part of the problem domain.
- These things **know** about their data: instance variables.
- These things **do** work on that data: methods.

Keys to OOP

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism



Object Oriented Programming (OOP)

Abstraction:

- Show only essential features of the application while hiding the details.
- In Java, **classes** provide **methods** to the outside world that allow access and use of the data variables.
- The data, however, remains hidden.

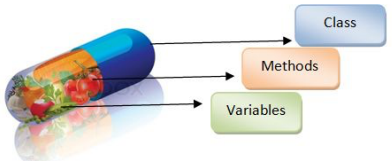
What happens when you press the power button?

Do you really need to know?

Object Oriented Programming (OOP)

Encapsulation:

- All about binding data variables and methods (that act on that data) together in a **class**.
- Classes provide that data representation.
- A Client uses the data type as a black box.
- The API specifies the contract between client and class.



You don't need to know how a data type is implemented in order to use it!

Wentworth
Computing & Data Science

Object Oriented Programming (OOP)

Client

API

- Volume
- Change Channel
- Adjust Picture

Implementation

- LCD Monitor
- 120Hz Refresh
- Wall Mountable

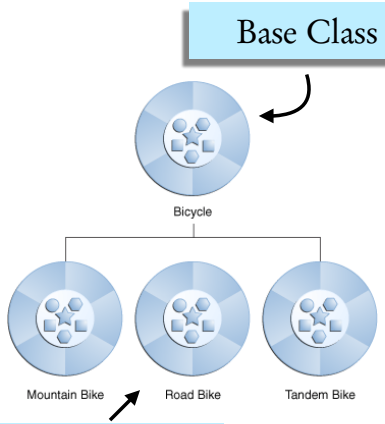
Client only needs to know how to use the API.

Implementation needs to know what API to implement.

Object Oriented Programming (OOP)

Inheritance:

- A way to reuse code over and over again.
- Start with some **base class** that contains data and methods.
- A **derived class** inherits from the base class and is able to use any* methods or variables defined in the base class.



Wentworth
Computing & Data Science

Object Oriented Programming (OOP)

Polymorphism:

- Lets us process objects differently depending on their data type.
- Essentially, one method with multiple implementations that is decided at run-time.
- We can define a generic interface which can hide multiple underlying implementations!

One implementation for the GPU

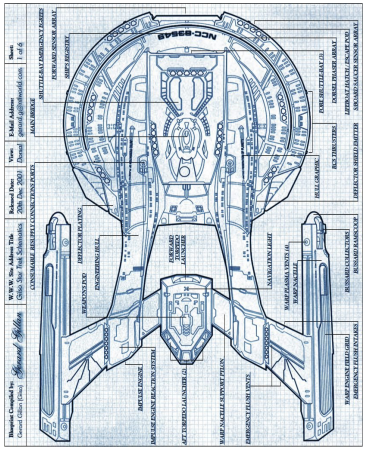
And one for the CPU

Wentworth
Computing & Data Science

Creating Classes

Classes are **blueprints** for objects

- They define the data fields and methods that an object of that type can have.
- Creating an object from a class blueprint is called **instantiation**.
- You can use **instance** or **object** interchangeably.



Creating Classes

Basic Class Diagram

```
Name: Circle

Fields:
    radius

Methods:
    getArea
    setRadius
```

Class Name

Data Fields

Class Methods

Instantiated Object of the Circle Class

```
Object1

Fields:
    radius=1
```

```
Object2

Fields:
    radius=5
```

```
Object3

Fields:
    radius=25
```

Anatomy of a Class

```
class Circle{


    //Circle Constructor
    constructor(){
      this.radius=1;
    }

    //Return the area of the circle
    getArea(){
        return this.radius * this.radius *
                                    Math.PI;
    }

    //Give a new radius to this circle
    setRadius(newRadius){
        this.radius = newRadius;
    }
}
```

Constructor With Fields

Methods

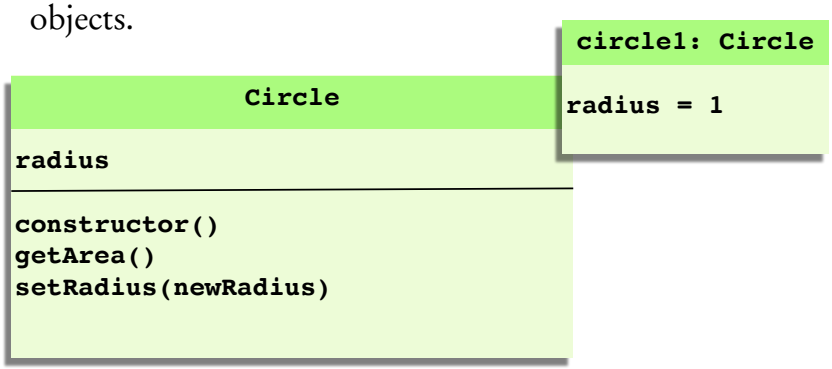
The `this` Keyword

- There is a special keyword that you use when you need to refer to the "current" object within a class.
- **`this`**
- It is particularly useful to clarify which object you are accessing where there are multiple objects
- You will use it all the time in JavaScript.

This is probably the one thing I dislike the most about JavaScript (and Python).

UML Diagrams

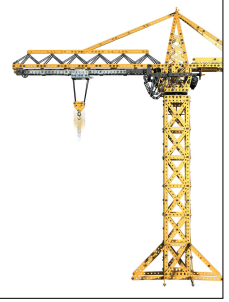
- Provides a standard way to represent classes and objects.



Let's Talk About Constructors

- Invoke a constructor to create an object using the **new** keyword.
- The constructor **must** have the name "**constructor**".
- They don't return anything.
- When the constructor is invoked with **new**, it plays the role of initializing objects.

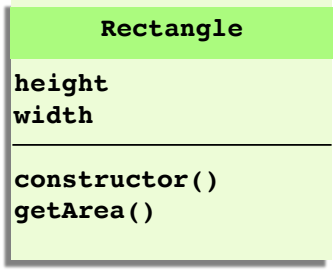
```
circle1 = new Circle();
```



Exercise

Create a Rectangle Class

- Write the `Rectangle` class in a new file in your project.
- Include two double fields, `height` and `width`.
- Add a constructor that sets the height and width.
- Include a function, `getArea()`, that returns the area of the rectangle.



Creating a new file in your project requires that it be added to **`index.html`** to be used.

Answer: The Rectangle Class

```
class Rectangle{

    //Circle Constructor
    constructor(width,height){
      this.width=width;
      this.height=height
    }

    //Return the area of the circle
    getArea(){
        return this.width*this.height;
    }

}
```

Object References

- Objects are accessed via the object reference.
- An object's data and methods are accessed via the dot(`.`) operator.
- The dot operator follows the reference to the object.
- The variable '**`radius`**' is known as an instance variable because it is dependent on the specific instance.

Instantiation and assignment

```
circle1 = new Circle();
```

Accessing the Radius Variable

```
circle1.radius
```


Using the Rectangle Class

In the index.html file, add your new JavaScript file like this:

```
<script src="sketch.js"></script>
<script src="Rectangle.js"></script>
```

This gives your sketch (main program) access to the code that you wrote in your new file.

Now, create a rectangle object in your main sketch and run the **`getArea()`** function. Print out the resulting value.

Update our Bouncing Ball

Let's update our bouncing ball program to use an object for the ball instead of having all the code in the main file, setup, and draw.

Also, let's add a "kick" that adds some random velocity when we click the mouse.

We'll need to lookup how to detect a mouse click.

Bubbles!

Let's make a Bubble class.

Each bubble will be a circle with an outline only.

They move by randomly jittering around on the screen.

They'll have two methods, **`move()`** and **`show()`** that we will call from **`draw()`**.

Wrap up

- A class defines a complex variable type
  - Contains its own data fields and functions that are only for use with objects of that class
- There are many predefined classes in JavaScript.
- You can also define your own classes
  - Often done to represent an entity in your program that requires more than one variable
- This is just the beginning of object oriented (OO) software development