

Generative Design In Processing

A simple guide to creating an Autonomous Agent

www.github.com/creativetechnologylab

Structural Overview

```
class Ant{  
}
```

```
void setup() {  
}
```

```
void draw() {  
}
```

```
void mousePressed() {  
}
```

```
void keyPressed() {  
}
```

Introduction

```
void setup() {  
    size(600, 600);  
    background(255);  
}
```

```
void draw() {}
```

Data Type

```
void draw() {  
    circle(xPosition, yPosition, 50);  
}
```

Data Type

```
float xPositon,yPosition;
```

```
void setup() {
```

```
// -- previous code -- //
```

```
}
```

int : 0 10 1 3

float: 3.14

string: "hello world"

Data Type

```
void setup() {  
    size(600, 600);  
    background(255);  
    xPos = width / 2;  
    yPos = height/2;  
}  
  
void draw() {  
    circle(xPos, yPos, 50);  
}
```

Object Orientated Programming

```
class Ant {  
    Ant() {}  
}
```

the class is the
blueprint and
instruction set

Object Orientated Programming

```
class Ant {  
    Ant() {  
    }  
    void move() {}  
  
    void display() {}  
}
```

These methods, `move` and `display`, are like actions the Ant can perform. `move` will change the position, and `display` will show it on the canvas.

Object Orientated Programming

```
class Ant {  
    float x, y;  
    Ant() {  
    }  
    void move() {}  
  
    void display() {}  
}
```

x,y: “member variables” are the
Ants “long-term memory”

Object Orientated Programming

```
class Ant {  
    float x, y;  
    Ant(float inputX, float inputY) {  
        x = inputX;  
        y = inputY;  
    }  
}
```

inputX, inputY:
properties that an individual ant can have. These variables are the ant's short term memory that we pass to the long-term memory

Methods: Move and Display

```
void move() {  
    x += random(-3,3);  
    y += random(-3,3);  
}
```

```
void display() {  
    fill(255,0,0);  
    circle(x, y, 20);  
}
```



+= will update the x
variable

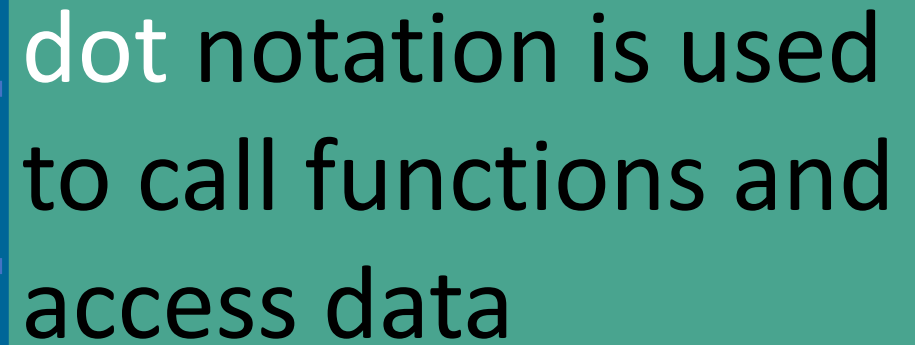
```
Ant jim;  
float xPositon,yPosition;  
  
void setup() {  
    size(600, 600);  
    background(255);  
    xPositon = width/2;  
    yPositon = height/2;  
    jim = new Ant(xPositon,yPosition);  
}
```

Ant is a type like **int** and **float**.
A class is a blue print for a
type of object.

new is a key word used to
construct a new **Ant**.

```
class Ant {  
    Ant() {} ← constructor  
}
```

```
void draw() {  
    jim.display();  
    jim.move();  
}
```



dot notation is used
to call functions and
access data

Data Structures and Loops

...previous variable code...

```
Ant jim;  
Ant[] antHill;
```



a **type Ant** data structure
called antHill

```
void setup() {  
  size(600, 600);  
  background(255);
```

... previous setup code ...

```
  antHill = new Ant[10];  
}
```



this antHill will hold 10
ants

Data Structures and Loops

```
void setup() {
```

```
    ...setup code...
```

```
    for (int i = 0; i < 10; i+=1) {
```

```
        antHill[i] = new Ant(xPosition, yPosition);
```


```
    }
```

```
}
```


Data Structures and Loops

```
void setup() {  
  size(600, 600);  
  background(255);  
  antHill = new Ant[10];  
  for (int i = 0; i < 10; i+=1) {  
    antHill[i] = new Ant(xPosition, yPosition);  
  }  
}
```

3 parts of a loop:
the counter start
the counter end
the counter step



count into the data
structure



i++ instead of
i+=1

antHill.length
instead of 10

Data Structures and Loops

```
void draw() {  
    for (Ant ants : antHill) {  
        ants.move();  
        ants.display();  
    }  
}
```



for **type** ants in
antHill

quicker loop syntax
but you cannot
address individual
items.

The Ant Class: Updating Movement

```
void move() {  
    float choice = random();  
    ... previous move code ...  
}
```

The Ant Class: Updating Movement

```
void move() {  
    ... previous move code ...  
    if (choice < 0.5) {  
        x += random(-3,3)  
    } else {  
        y += random(-3,3)  
    }  
}
```

The Ant Class: Updating Movement

```
void move() {  
    ... previous move code ...  
    if(x > width) x = 0;  
    if(y > height) y = 0;  
    if(x < 0) x = width;  
    if(y < 0) y = height;  
}
```

Methods: Updating Display

```
class Ant {  
    ... previous variable code ...  
    float size;  
    color bodyColor;  
    Ant(float inputX, float inputY) {  
        ... previous Ant code ...  
        bodyColor = color(255,0,0);  
        size = 10;  
    }  
}
```

```
void display() {  
    fill(bodyColor);  
    circle(x, y, size);  
}
```

Methods: Events

```
void setup() {  
    ...previous setup code...  
}  
  
void draw() {  
    ....previous draw code ....  
}
```

```
void mousePressed() {  
    for (Ant ants : antHill) {  
        float r = random(128, 255);  
        float g = random(198, 255);  
        float b = random(64, 195);  
        float size = random(1,10);  
        ants.bodyColor = color(r,g,b);  
        ants.size = size;  
    }  
}
```

Methods: Events

```
class Ant{  
    ...previous class code...  
}  
  
void setup() {  
    ...previous setup code...  
}  
  
void draw() {  
    ....previous draw code ....  
}
```

```
void mousePressed() {  
    ....previous mouse code ....  
}  
  
void keyPressed() {  
    if (key == 's' || key == 'S') {  
        saveFrame();  
        exit();  
    }  
}
```


Output

open the Processing
sketch folder
this project is
saved in

