

Day 4

```
Funct i o n s ( ) {
```

```
}
```

Agenda

1. What is a function?
2. Why do we use functions?
3. Structure of a function
4. The `map()` function

Functions

Like sentences.

To repeat with different words.

What is a function?

A function is a **set of organised commands used to perform a specific action.**

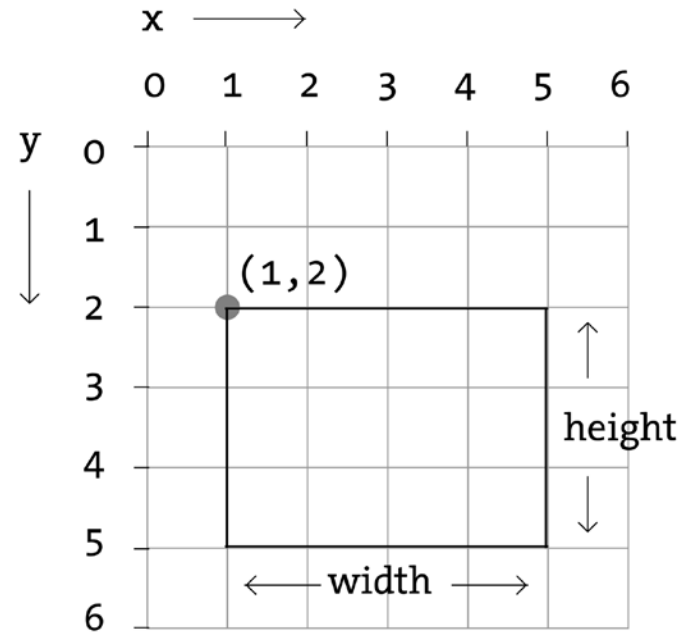
Examples:

- Draw rectangle
- Get the time
- Plan a vacation

```
Task {  
    Step to complete the task  
    Step to complete the task  
    Step to complete the task  
    Step to complete the task  
    Step to complete the task  
}
```

Why do we use functions?

- Reusability
- Organization
- Abstraction



Reusability

- Define **repeated tasks**
- Write **less code**

```
Main Program {  
    Take out mobile phone;  
    Turn it on;  
    Look at the time;  
    Go to class;  
    Give presentation;  
    Take out mobile phone;  
    Turn it on;  
    Look at the time;  
    Have lunch;  
    Take out mobile phone;  
    Turn it on;  
    Look at the time;  
    Go to buy a coffee;  
    Meet for group project;  
}
```

Reusability

- Create a function **outside of main code**
- **“Call” the function** as many times as you'd like

```
Main Program {  
    Get the time;  
    Go to class;  
    Give presentation;  
    Get the time;  
    Have lunch;  
    Get the time;  
    Go to buy a coffee;  
    Meet for group project;  
}
```

```
Get the time {  
    Take out mobile phone;  
    Turn it on;  
    Look at the time;  
}
```

Organization

- Break many lines of code into **smaller, digestible “building blocks”**
- Structure in a way that is **easy to read, review, and debug** for yourself and for others

```
Main Program { // plan a vacation
    Choose destination;
    Set dates;
    Establish budget;
    Read travel guides;
    Ask for recommendations;
    Compare prices;
    Create itinerary;
    Buy flights;
    Book accommodations;
    Make reservations;
    Apply for visa;
    Renew passport;
    Get vaccinations;
    Buy travel insurance;
}
```



```
Main Program {
```

```
    Decide;  
    Research;  
    Decide;  
    Research;  
    Book;  
    Prepare;
```

```
}
```

```
    Decide {
```

```
        Choose destination;  
        Set dates;  
        Establish budget;
```

```
    }
```

```
    Research {
```

```
        Read travel guides;  
        Ask for recommendations;  
        Compare prices;  
        Create itinerary;
```

```
    }
```

```
    Book {
```

```
        Buy flights;  
        Book accommodations;  
        Make reservations;
```

```
    }
```

```
    Prepare {
```

```
        Apply for visa;  
        Renew passport;  
        Get vaccinations;  
        Buy travel insurance;
```

```
    }
```

Abstraction

- Let's you carry out a task **without knowing the details of the implementation**

e.g.

Draw a rectangle without needing to know the steps to create each line

Shape

`createShape()`

`loadShape()`

`PShape`

2D Primitives

`arc()`

`ellipse()`

`line()`

`point()`

`quad()`

`rect()`

`triangle()`

Color

Setting

`background()`

`clear()`

`colorMode()`

`fill()`

`noFill()`

`noStroke()`

`stroke()`

Some of Processing's built-in functions

processing.org/reference

Structure

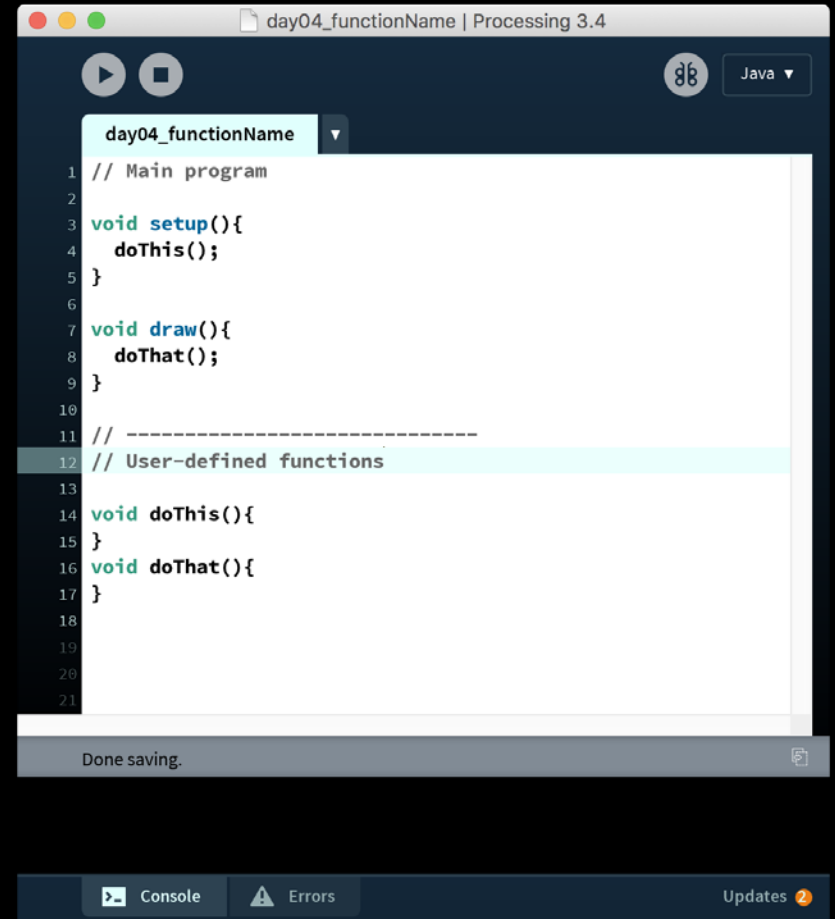
- **Parameters:** the “materials” we provide for the function
- **Body:** what we want the function to do
- **Return value:** what we want the function to give back to us

```
return-type function-name  
(function-parameters) {  
    // body  
    Local variables;  
    Variable definitions;  
    return-value;  
}
```

```
// example  
int sum(int a, int b) {  
    int result;  
    result = a + b;  
    return result;  
}
```

function-name

- Used to “call” the function elsewhere in the program
- Best practice: **be specific**; designate a name that **describes the action**
e.g. `sayMyName()` vs. `name()`



```
day04_functionName | Processing 3.4
// Main program
1 // Main program
2
3 void setup(){
4   doThis();
5 }
6
7 void draw(){
8   doThat();
9 }
10
11 // -----
12 // User-defined functions
13
14 void doThis(){
15 }
16 void doThat(){
17 }
18
19
20
21
```

Done saving.

Console Errors Updates 2

return-type +
return-value

- The **format of data** that the function will return
- The **return value** must be consistent with function's **return type**
e.g. `int` returns an integer, `bool` returns true or false

```
int functionName(){  
    // do this  
    // do that  
    // return value of type int  
}
```

```
bool functionName(){  
    // do this  
    // do that  
    // return value of type bool  
}
```

```
String functionName(){  
    // do this  
    // do that  
    // return value of type String  
}
```

return-type +

return-value

- If **no value** needs to be returned, use `void`

e.g. `setUp()`, `draw()`, and `mousePressed()`

```
void functionName() {  
    // do this  
    // do that  
    // exit  
}
```

```
String whatsMyName() {  
    return "Rihanna";  
}
```

```
// call the function
```

```
String myName = whatsMyName();  
println(myName);
```

```
// or
```

```
println(whatsMyName());
```

vs.

```
void sayMyName() {  
    println("Destiny's  
Child");  
}
```

```
// call the function
```

```
sayMyName();  
sayMyName();
```

function-parameters

- Are **values, and their types**, passed into a function (the “**materials**”)
- Gives the function **flexibility**
- A function can take **multiple** parameters, but
- **Not all** functions require parameters
e.g. `setUp()`, `draw()`

```
// built-in function
void rect(float posX, float posY,
float width, float height){

    // definition
}

// call the function
rect(10, 10, 20, 40);
rect(5.5, 7.5, 3.25, 5.25);
```


variables

- **Global variables:** defined in the main program. They can be used by any function.
- **Local variables:** defined within a function, including parameters. They can't be used outside that function.

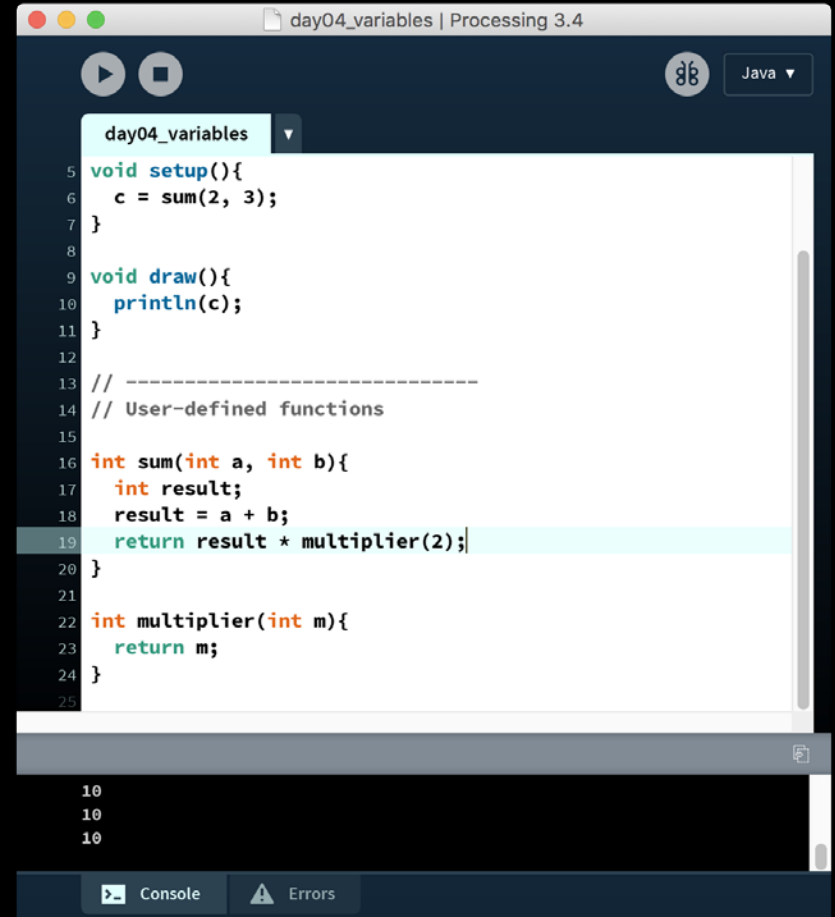
```
day04_variables | Processing 3.4
Java
day04_variables
1 // Main program
2
3 int c; // global variable
4
5 void setup(){
6   c = sum(2, 3);
7 }
8
9 void draw(){
10  println(c);
11  println(result);
12 }
13
14 // -----
15 // User-defined functions
16
17 int sum(int a, int b){ // parameters
18   int result; // local variable
19   result = a + b; // local definition
20   return result;
21 }
```

The variable "result" does not exist

Console Errors

variables

- A function can also be called **within another function**



```
day04_variables | Processing 3.4  
day04_variables  
5 void setup(){  
6   c = sum(2, 3);  
7 }  
8  
9 void draw(){  
10  println(c);  
11 }  
12  
13 // -----  
14 // User-defined functions  
15  
16 int sum(int a, int b){  
17   int result;  
18   result = a + b;  
19   return result * multiplier(2);  
20 }  
21  
22 int multiplier(int m){  
23   return m;  
24 }  
25
```

10
10
10

Console Errors

What's logic of this sequence of numbers?

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

What's logic of this sequence of numbers?

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

Fibonacci Numbers

$$X(n) = X(n-2) + X(n-1)$$

$$X(0) = 0, X(1) = 1$$

What's logic of this sequence of numbers?

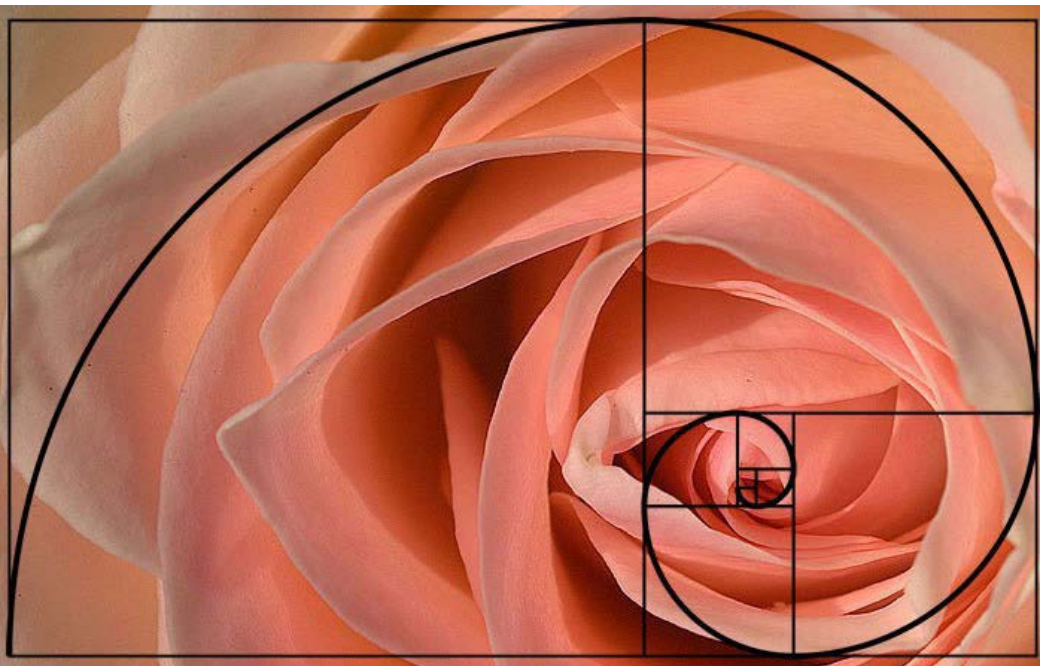
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

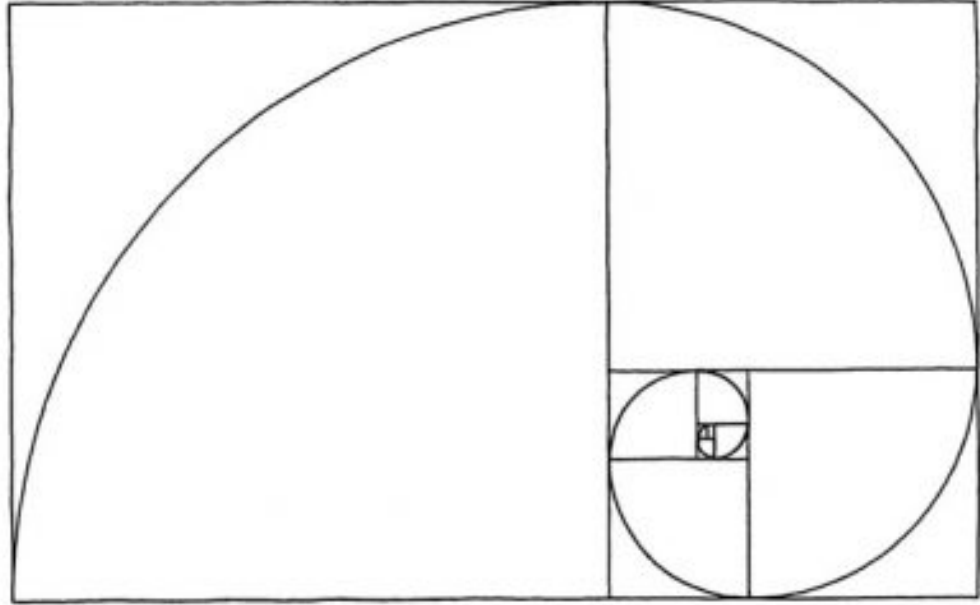
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

Fibonacci Numbers

$$X(n) = X(n-2) + X(n-1)$$

$$X(0) = 0, X(1) = 1$$

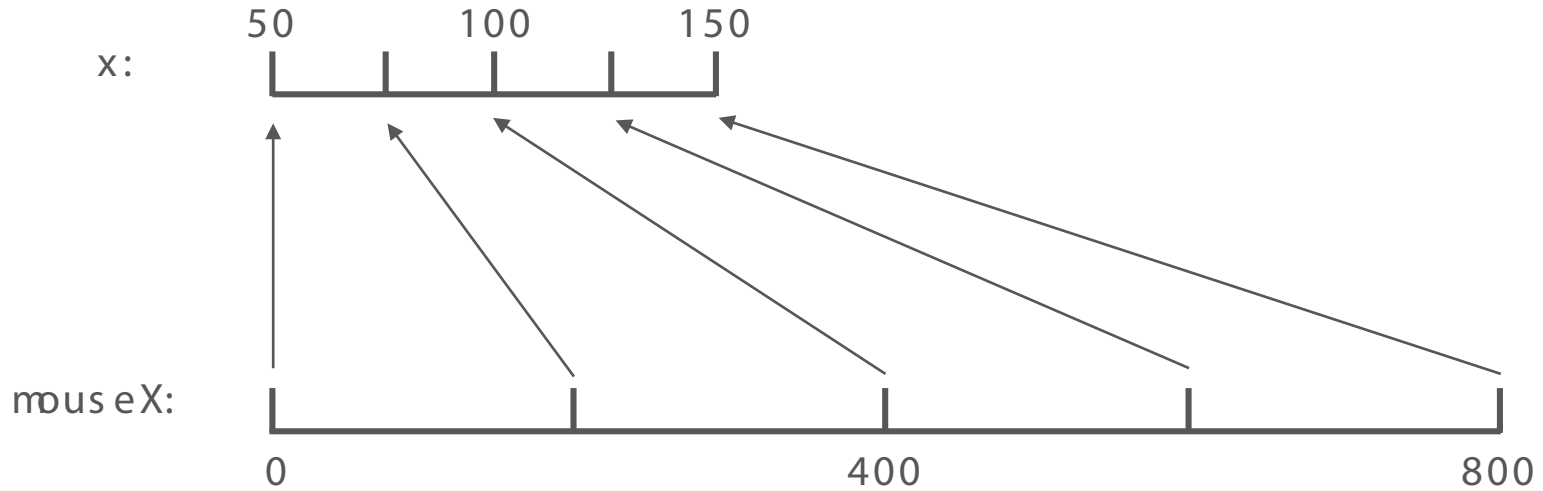




Br eak

map() function

```
map( value, start1, stop1, start2, stop2 );  
float x = map(mouseX, 0, 800, 50, 150);
```



Let ' s code.

Task

if mouse clicked draw a circle at mouse position;
as mouse moves towards the right edge of canvas, the radius grows
Scale the radius of the circle to be between 30 and 150, based on
the mouse position

Task

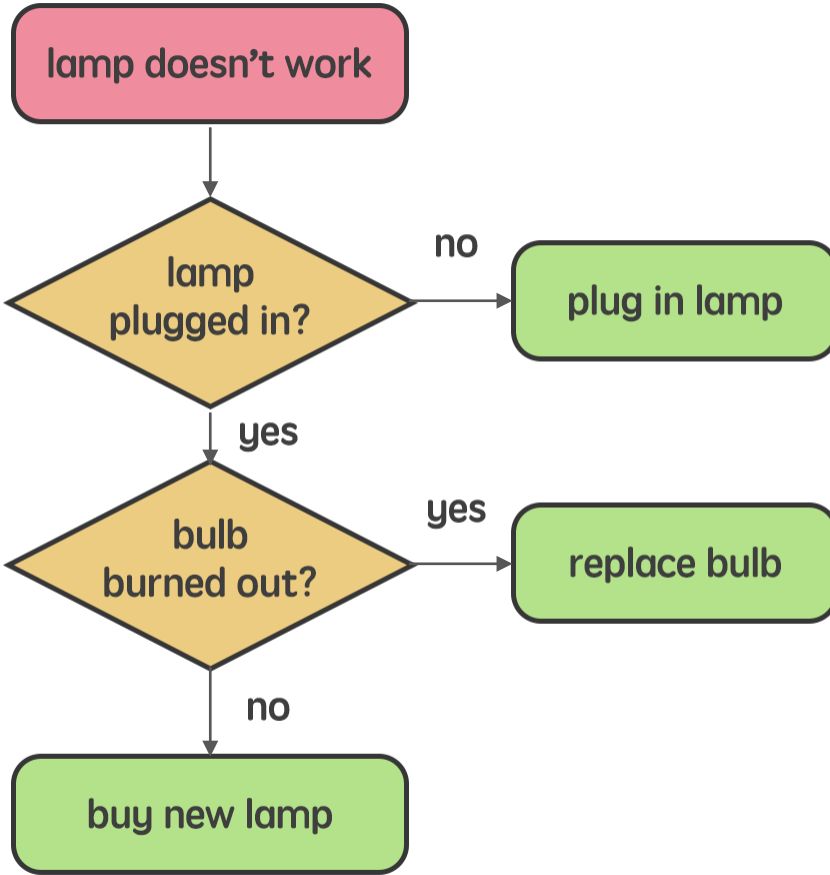
draw your flow chart first
check with your partner

Homework

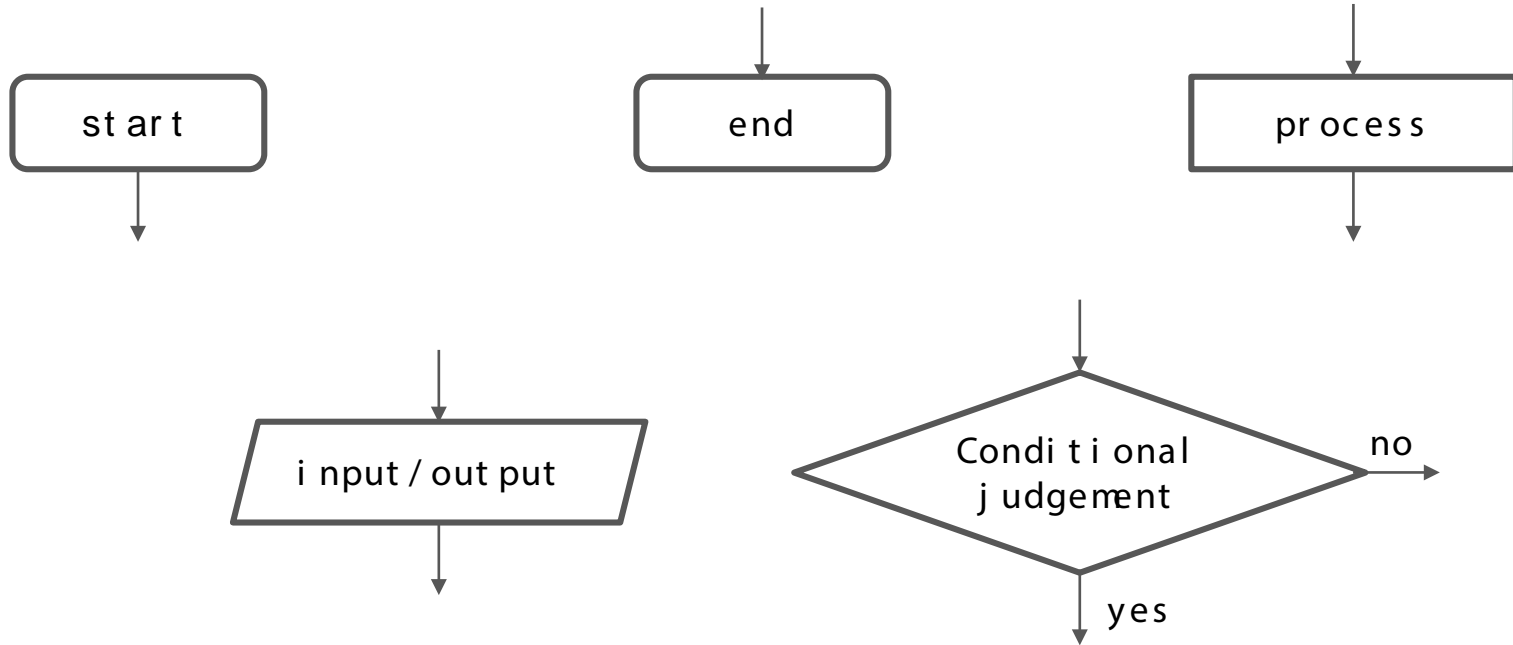
Continue working on text adventure

- Finish 3 stages
- Add a function
- Add pictures
- Make Slides

Flow Chart



Flow chart



It's your turn