# Mr C is getting dizzy

ESC101: Fundamentals of Computing

Purushottam Kar

# Lab Exam (Sun, 09 Sep)

- Morning exam (Wed, Thu batches)
  - 10:30 AM - 1:30 PM – starts 10:30 AM sharp
  - **CC-01**: B9, {B14 even roll numbers}
  - **CC-02**: B7, B10, B11
  - **CC-03**: B12
  - **MATH-LINUX**: B8, {B14 odd roll numbers}
- Go see your room during this week's lab
- Be there 15 minutes before your exam 10:15AM
- Cannot switch to afternoon session

# Lab Exam (Sun, 09 Sep)

- Afternoon exam (Mon, Tue batches)
  - 2 PM - 5 PM – starts 2 PM sharp
  - **CC-01**: B1, {B2 even roll numbers}
  - **CC-02**: B4, B5, B6
  - **CC-03**: B3
  - **MATH-LINUX**: B13, {B2 odd roll numbers}
- Go see your room during this week's lab
- Be there 15 minutes before your exam 1:45 PM
- Cannot switch to morning session

# Lab Exam (Sun, 09 Sep)

- Syllabus – till loops (no arrays)
- Exam will be like labs
- Marks for passing test cases
- Marks for writing clean indented code, proper variable names, a few comments on what steps are being performed
- Illegible code passing all test cases may get only a fraction of total marks

# Advanced Track

- Offers will be made tonight – based on performance in labs in week 2-5, minor quiz in week 2-4 and major quiz
- No bonus questions counted in deciding AT offers – will be counted when computing final grade
- Students will have 2-3 days to either agree to switch or decline the offer
- Staying silent will mean declining offer

# The for loop




भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

# The for loop



## General form of a for loop

# The for loop



General form of a for loop

```
for(init_expr; stopping_expr; update_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

# The for loop



General form of a for loop

```
for(init_expr; stopping_expr; update_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

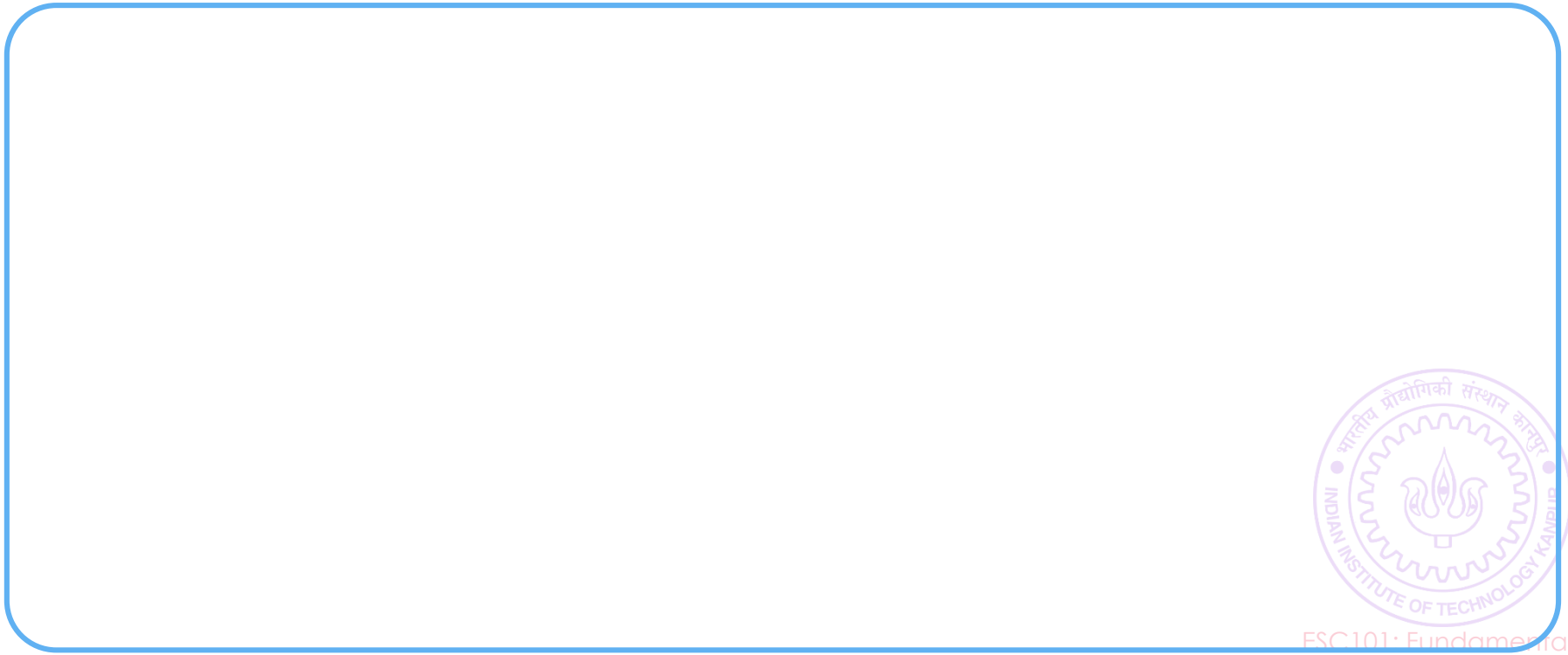
# The for loop



General form of a for loop

```
for(init_expr; stopping_expr; update_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**



# The for loop



General form of a for loop

```
for(init_expr; stopping_expr; update_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First do what is told in initialization expression

# The for loop



General form of a for loop

```
for(init_expr; stopping_expr; update_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First do what is told in initialization expression

# The for loop



General form of a for loop

```
for(init_expr; stopping_expr; update_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First do what is told in initialization expression
2. Then check the stopping expression

# The for loop



General form of a for loop

```
for(init_expr; stopping_expr; update_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First do what is told in initialization expression
2. Then check the stopping expression

# The for loop



General form of a for loop

```
for(init_expr; stopping_expr; update_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First do what is told in initialization expression
2. Then check the stopping expression
3. If stopping expression is true

# The for loop



General form of a for loop

```
for(init_expr; stopping_expr; update_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First do what is told in initialization expression
2. Then check the stopping expression
3. If stopping expression is true
   Execute all statements inside braces

# The for loop



General form of a for loop

```
for(init_expr; stopping_expr; update_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First do what is told in initialization expression
2. Then check the stopping expression
3. If stopping expression is true
   Execute all statements inside braces

# The for loop



General form of a for loop

```
for(init_expr; stopping_expr; update_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First do what is told in initialization expression
2. Then check the stopping expression
3. If stopping expression is true
   - Execute all statements inside braces
   - Execute update expression

# The for loop



General form of a for loop

```
for(init_expr; stopping_expr; update_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First do what is told in initialization expression
2. Then check the stopping expression
3. If stopping expression is true
   - Execute all statements inside braces
   - Execute update expression

# The for loop



General form of a for loop

```
for(init_expr; stopping_expr; update_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First do what is told in initialization expression
2. Then check the stopping expression
3. If stopping expression is true
   - Execute all statements inside braces
   - Execute update expression
   - Go back to step 2

# The for loop

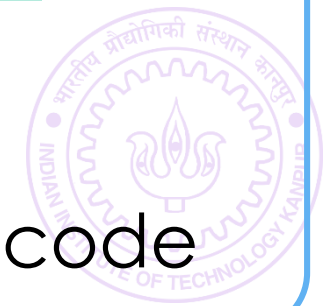


General form of a for loop

```
for(init_expr; stopping_expr; update_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First do what is told in initialization expression
2. Then check the stopping expression
3. If stopping expression is true
   - Execute all statements inside braces
   - Execute update expression
   - Go back to step 2
   - Else stop looping and execute rest of code



# The for loop

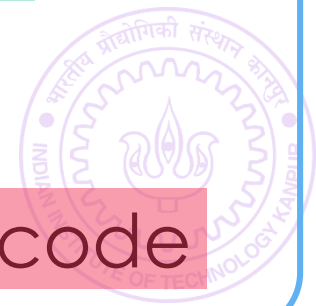


General form of a for loop

```
for(init_expr; stopping_expr; update_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First do what is told in initialization expression
2. Then check the stopping expression
3. If stopping expression is true
   - Execute all statements inside braces
   - Execute update expression
   - Go back to step 2
   - Else stop looping and execute rest of code



# The for loop

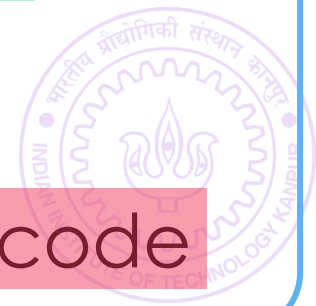


General form of a for loop

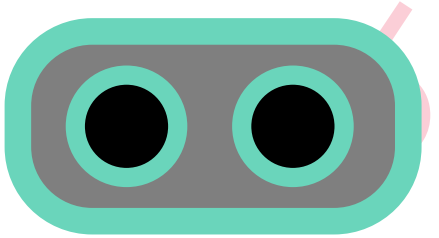
```
for(init_expr; stopping_expr; update_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First do what is told in initialization expression
2. Then check the stopping expression
3. If stopping expression is true
   - Execute all statements inside braces
   - Execute update expression
   - Go back to step 2
   - Else stop looping and execute rest of code



# The for loop



General form of a for loop

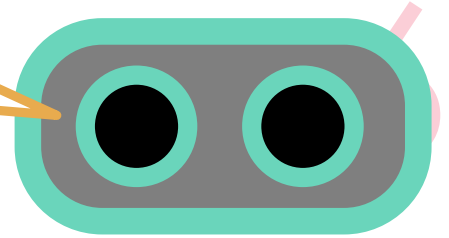
```
for(init_expr; stopping_expr; update_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First do what is told in initialization expression
2. Then check the stopping expression
3. If stopping expression is true
   - Execute all statements inside braces
   - Execute update expression
   - Go back to step 2
   - Else stop looping and execute rest of code

# The for loop

Brackets essential if you want me to do many things while looping



General form of a for loop

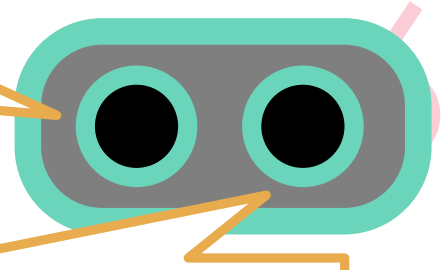
```
for(init_expr; stopping_expr; update_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First do what is told in initialization expression
2. Then check the stopping expression
3. If stopping expression is true
   - Execute all statements inside braces
   - Execute update expression
   - Go back to step 2
   - Else stop looping and execute rest of code

# The for loop

Brackets essential if you want me to do many things while looping



Each time I execute the statements inside the braces – called one *iteration* of the loop

General form of a for loop

```
for(init_expr; stopping_expr; update_
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First do what is told in initialization expression
2. Then check the stopping expression
3. If stopping expression is true
   - Execute all statements inside braces
   - Execute update expression
   - Go back to step 2
   - Else stop looping and execute rest of code

# The while loop

# The while loop



General form of a while loop

# The while loop



General form of a while loop

```
while(stopping_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

# The while loop



General form of a while loop

```
while(stopping_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

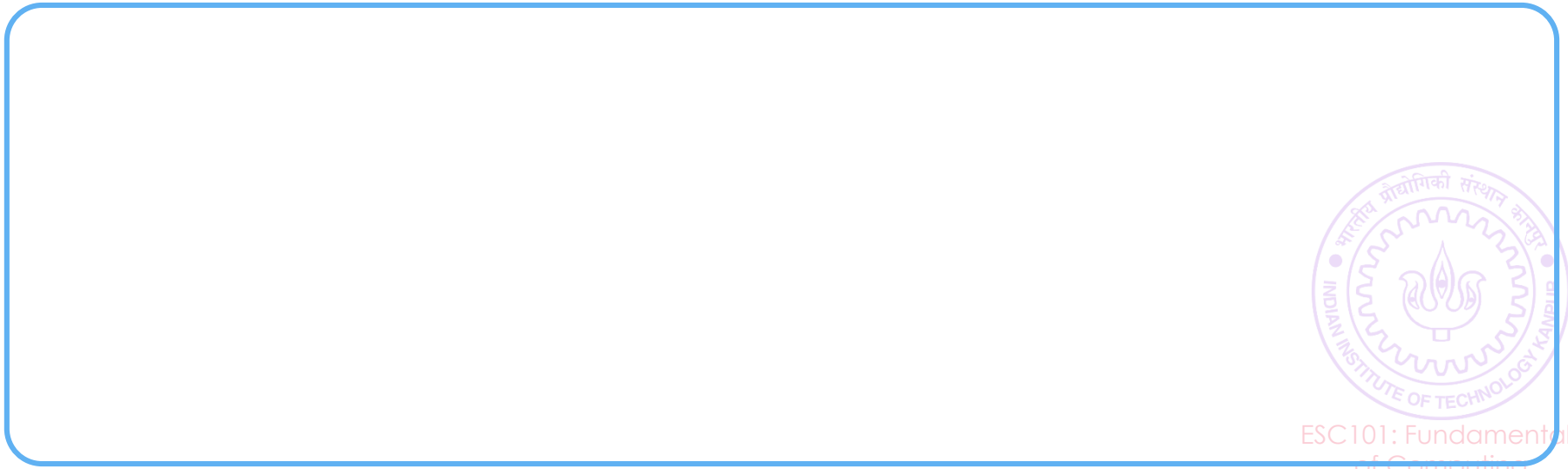
# The while loop



General form of a while loop

```
while(stopping_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**



# The while loop



General form of a while loop

```
while(stopping_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First check the stopping expression

# The while loop



General form of a while loop

```
while(stopping_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First check the stopping expression

# The while loop



General form of a while loop

```
while(stopping_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First check the stopping expression
2. If stopping expression is true

# The while loop



General form of a while loop

```
while(stopping_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First check the stopping expression
2. If stopping expression is true
   Execute all statements inside braces

# The while loop



General form of a while loop

```
while(stopping_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First check the stopping expression
2. If stopping expression is true
   Execute all statements inside braces

# The while loop



General form of a while loop

```
while(stopping_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First check the stopping expression
2. If stopping expression is true
   Execute all statements inside braces
   Go back to step 2

# The while loop

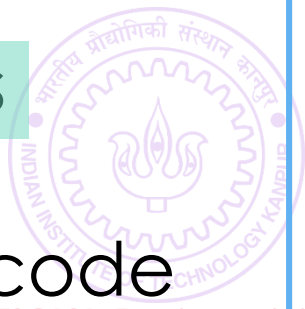


General form of a while loop

```
while(stopping_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First check the stopping expression
2. If stopping expression is true
   - Execute all statements inside braces
   - Go back to step 2
   - Else stop looping and execute rest of code



# The while loop

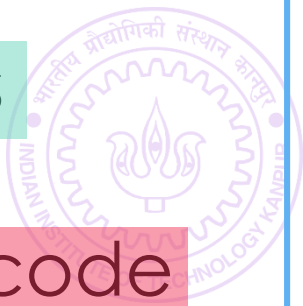


General form of a while loop

```
while(stopping_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First check the stopping expression
2. If stopping expression is true
   - Execute all statements inside braces
   - Go back to step 2
   - Else stop looping and execute rest of code



# The while loop

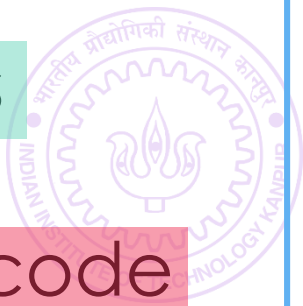


General form of a while loop

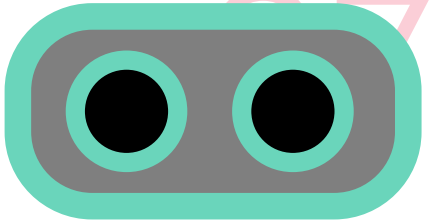
```
while(stopping_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First check the stopping expression
2. If stopping expression is true
   - Execute all statements inside braces
   - Go back to step 2
   - Else stop looping and execute rest of code



# The while loop



General form of a while loop

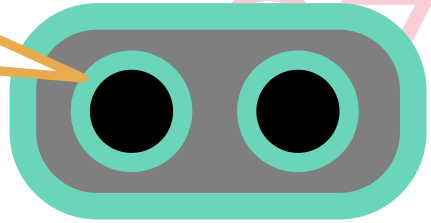
```
while(stopping_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First check the stopping expression
2. If stopping expression is true
   Execute all statements inside braces
   Go back to step 2
   Else stop looping and execute rest of code

# The while loop

Brackets essential if you want me to do many things while looping



General form of a while loop

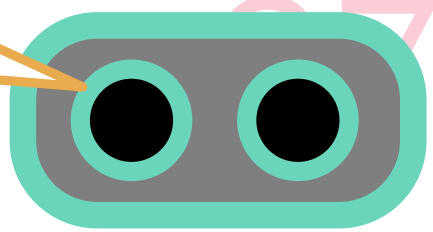
```
while(stopping_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First check the stopping expression
2. If stopping expression is true
   - Execute all statements inside braces
   - Go back to step 2
   - Else stop looping and execute rest of code

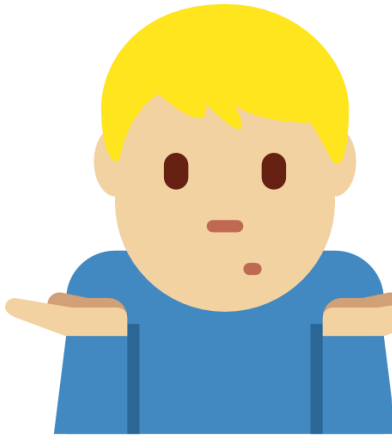
# The while loop

Brackets essential if you want me to do many things while looping



General form of a while loop

```
while(stopping_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

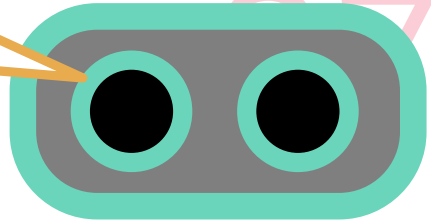


**How we usually speak to a human**

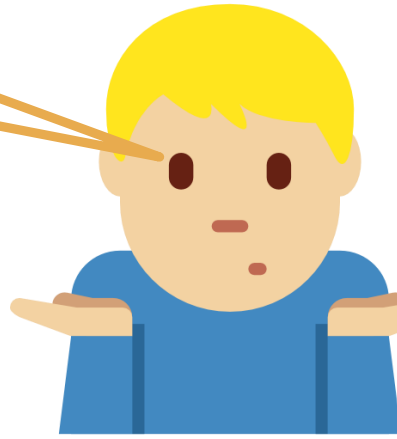
1. First check the stopping expression
2. If stopping expression is true
   - Execute all statements inside braces
   - Go back to step 2
   - Else stop looping and execute rest of code

# The while loop

Brackets essential if you want me to do many things while looping



So what is the difference between for and while?



General form of a while loop

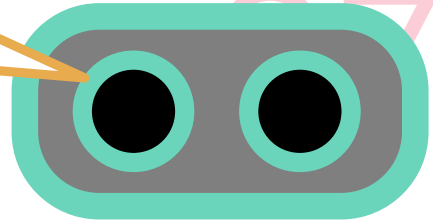
```
while(stopping_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

**How we usually speak to a human**

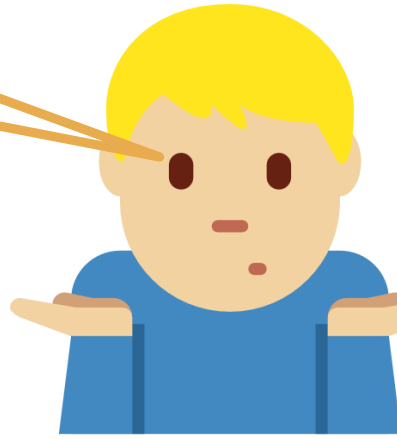
1. First check the stopping expression
2. If stopping expression is true
   Execute all statements inside braces
   Go back to step 2
   Else stop looping and execute rest of code

# The while loop

Brackets essential if you want me to do many things while looping

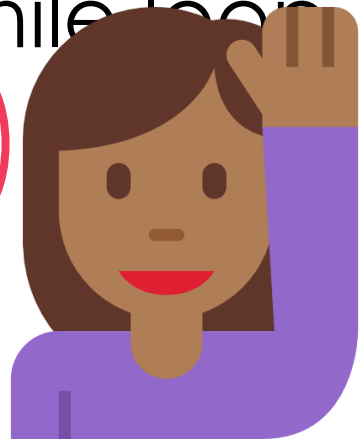


So what is the difference between for and while?



General form of a while loop

```
while(stopping_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```



**How we usually speak to a human**

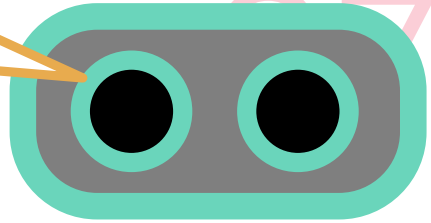
1. First check the stopping expression
2. If stopping expression is true
   - Execute all statements inside braces
   - Go back to step 2
   - Else stop looping and execute rest of code

# The while loop

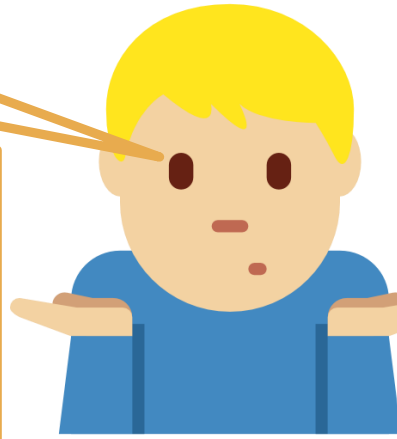
General form of a while loop

```
while(stopping_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

Brackets essential if you want me to do many things while looping



So what is the difference between for and while?



In general not much – it is a matter of style. Use while when you don't know how many iterations will loop run

**How we usually speak to a human**

1. First check the stopping expression
2. If stopping expression is true
   - Execute all statements inside braces
   - Go back to step 2
   - Else stop looping and execute rest of code

# The do-while loop

# The do-while loop



General form of a do-while loop

# The do-while loop



General form of a do-while loop

```
do{
    statement1;
    statement2;
    ...
}while(stopping_expr);
statement3;
statement4;
...
```

# The do-while loop



General form of a do-while loop

```
do{
    statement1;
    statement2;
    ...
}while(stopping_expr);
statement3;
statement4;
...
```

**How we usually speak to a human**

# The do-while loop



General form of a do-while loop

```
do{
    statement1;
    statement2;
    ...
}while(stopping_expr);
statement3;
statement4;
...
```

**How we usually speak to a human**

# The do-while loop



General form of a do-while loop

```
do{
    statement1;
    statement2;
    ...
}while(stopping_expr);
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First execute statements inside braces

# The do-while loop



General form of a do-while loop

```
do{
    statement1;
    statement2;
    ...
}while(stopping_expr);
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First execute statements inside braces

# The do-while loop



General form of a do-while loop

```
do{
    statement1;
    statement2;
    ...
}while(stopping_expr);
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First execute statements inside braces
2. Then check stopping criterion

# The do-while loop



General form of a do-while loop

```
do{
    statement1;
    statement2;
    ...
}while(stopping_expr);
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First execute statements inside braces
2. Then check stopping criterion

# The do-while loop



General form of a do-while loop

```
do{
    statement1;
    statement2;
    ...
}while(stopping_expr);
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First execute statements inside braces
2. Then check stopping criterion

2. If stopping expression is true

# The do-while loop



General form of a do-while loop

```
do{
    statement1;
    statement2;
    ...
}while(stopping_expr);
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First execute statements inside braces
2. Then check stopping criterion

2. If stopping expression is true
   Execute all statements inside braces

# The do-while loop



General form of a do-while loop

```
do{
    statement1;
    statement2;
    ...
}while(stopping_expr);
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First execute statements inside braces
2. Then check stopping criterion

2. If stopping expression is true
   Execute all statements inside braces

# The do-while loop



General form of a do-while loop

```
do{
    statement1;
    statement2;
    ...
}while(stopping_expr);
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First execute statements inside braces
2. Then check stopping criterion

2. If stopping expression is true
   Execute all statements inside braces
   Go back to step 2

# The do-while loop



General form of a do-while loop

```
do{
    statement1;
    statement2;
    ...
}while(stopping_expr);
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First execute statements inside braces
2. Then check stopping criterion

2. If stopping expression is true
   Execute all statements inside braces
   Go back to step 2
   Else stop looping, execute rest of code

# The do-while loop



General form of a do-while loop

```
do{
    statement1;
    statement2;
    ...
}while(stopping_expr);
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First execute statements inside braces
2. Then check stopping criterion

2. If stopping expression is true
   Execute all statements inside braces
   Go back to step 2
   Else stop looping, execute rest of code

# The do-while loop



General form of a do-while loop

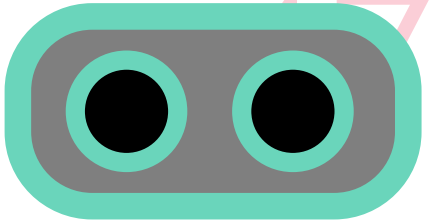
```
do{
    statement1;
    statement2;
    ...
}while(stopping_expr);
statement3;
statement4;
...
```

**How we usually speak to a human**

1. First execute statements inside braces
2. Then check stopping criterion

2. If stopping expression is true
   Execute all statements inside braces
   Go back to step 2
   Else stop looping, execute rest of code

# The do-while loop



General form of a do-while loop

```
do{
    statement1;
    statement2;
    ...
}while(stopping_expr);
statement3;
statement4;
...
```

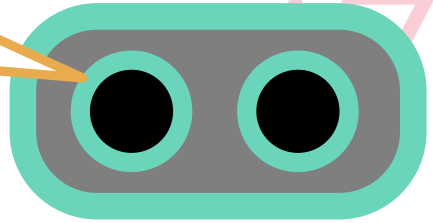
**How we usually speak to a human**

1. First execute statements inside braces
2. Then check stopping criterion

2. If stopping expression is true
   Execute all statements inside braces
   Go back to step 2
   Else stop looping, execute rest of code

# The do-while lo

Brackets essential if you want me to do many things while looping



General form of a do-while loop

```
do{
    statement1;
    statement2;
    ...
}while(stopping_expr);
statement3;
statement4;
...
```

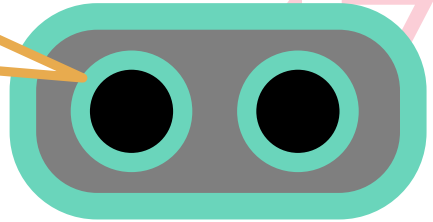
**How we usually speak to a human**

1. First execute statements inside braces
2. Then check stopping criterion

2. If stopping expression is true
   Execute all statements inside braces
   Go back to step 2
   Else stop looping, execute rest of code

# The do-while lo

Brackets essential if you want me to do many things while looping



General form of a do-while loop

```
do{
    statement1;
    statement2;
    ...
}while(stopping_expr);
statement3;
statement4;
...
```

**How we usually speak to a human**

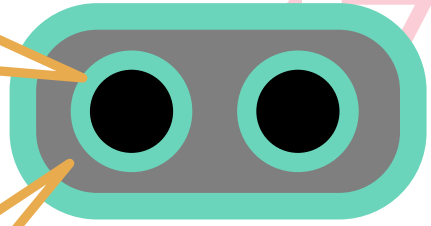
1. First execute statements inside braces
2. Then check stopping criterion

2. If stopping expression is true
   Execute all statements inside braces
   Go back to step 2
   Else stop looping, execute rest of code

# The do-while lo

Brackets essential if you want me to do many things while looping

Notice additional semi-colon



General form of a do-while loop

```
do{
    statement1;
    statement2;
    ...
}while(stopping_expr);
statement3;
statement4;
...
```

**How we usually speak to a human**

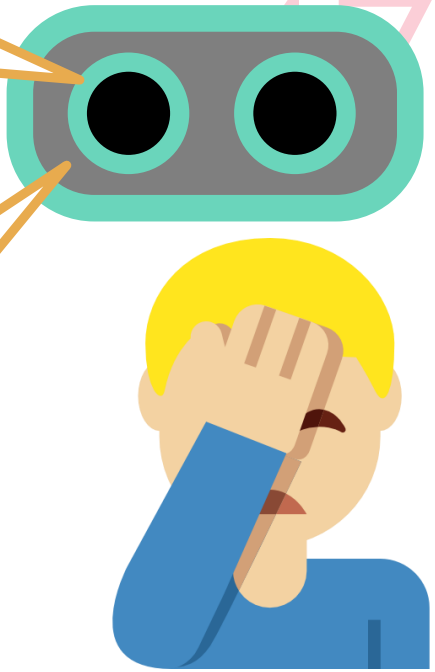
1. First execute statements inside braces
2. Then check stopping criterion

2. If stopping expression is true
   Execute all statements inside braces
   Go back to step 2
   Else stop looping, execute rest of code

# The do-while lo

Brackets essential if you want me to do many things while looping

Notice additional semi-colon



General form of a do-while loop

```
do{
    statement1;
    statement2;
    ...
}while(stopping_expr);
statement3;
statement4;
...
```

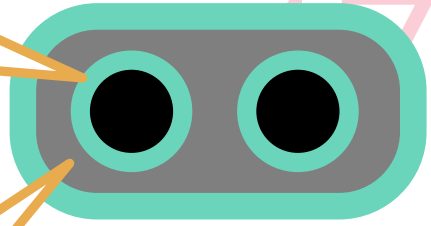
**How we usually speak to a human**

1. First execute statements inside braces
2. Then check stopping criterion

2. If stopping expression is true
   Execute all statements inside braces
   Go back to step 2
   Else stop looping, execute rest of code

# The do-while lo

Brackets essential if you want me to do many things while looping



Notice additional semi-colon

Yet another minor quiz question

General form of a do-while loop

```
do{
    statement1;
    statement2;
    ...
}while(stopping_expr);
statement3;
statement4;
...
```

**How we usually speak to a human**

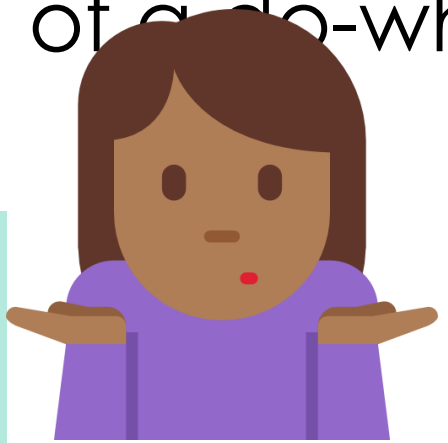
1. First execute statements inside braces
2. Then check stopping criterion

2. If stopping expression is true
   Execute all statements inside braces
   Go back to step 2
   Else stop looping, execute rest of code

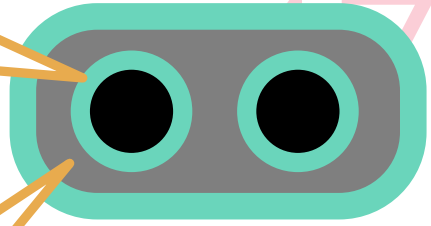
# The do-while lo

General form of a do-while loop

```
do{
    statement1;
    statement2;
    ...
}while(stopping_expr);
statement3;
statement4;
...
```



Brackets essential if you want me to do many things while looping



Notice additional semi-colon

Yet another minor quiz question

**How we usually speak to a human**

1. First execute statements inside braces
2. Then check stopping criterion

2. If stopping expression is true
   Execute all statements inside braces
   Go back to step 2
   Else stop looping, execute rest of code

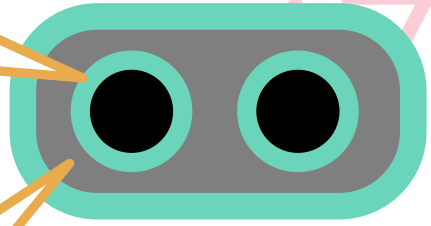
# The do-while lo

General form of a do-while loop

```
do{
    statement1;
    statement2;
    ...
}while(stopping_expr);
statement3;
statement4;
...
```

When to use do-while instead of while?

Brackets essential if you want me to do many things while looping



Notice additional semi-colon

Yet another minor quiz question

**How we usually speak to a human**

1. First execute statements inside braces
2. Then check stopping criterion

2. If stopping expression is true
   Execute all statements inside braces
   Go back to step 2
   Else stop looping, execute rest of code

# The use of do-while

# The use of do-while



The do-while loop is executed at least once

# The use of do-while



The do-while loop is executed at least once

Recall: *read integers till you read the number -1and …*

# The use of do-while



The do-while loop is executed at least once

Recall: *read integers till you read the number -1and …*

```
int num, sum = 0;
scanf(“%d”, &num);
while(num != -1){
    sum += num;
    scanf(“%d”, &num);
}
printf("%d",sum);
```

# The use of do-while



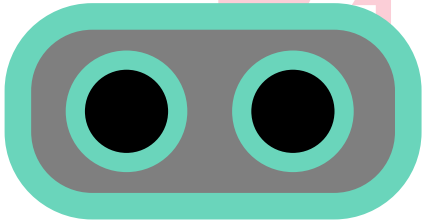
The do-while loop is executed at least once

Recall: *read integers till you read the number -1and …*

```
int num, sum = 0;
scanf(“%d”, &num);
while(num != -1){
    sum += num;
    scanf(“%d”, &num);
}
printf("%d",sum);
```

```
int num, sum = 0;
do{
    scanf(“%d”, &num);
    if(num != -1)
        sum += num;
}while(num != -1);
printf("%d",sum);
```

# The use of do-while



The do-while loop is executed at least once

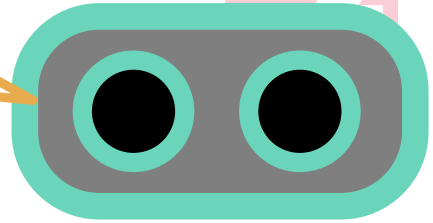
Recall: *read integers till you read the number -1and …*

```
int num, sum = 0;
scanf(“%d”, &num);
while(num != -1){
    sum += num;
    scanf(“%d”, &num);
}
printf("%d",sum);
```

```
int num, sum = 0;
do{
    scanf(“%d”, &num);
    if(num != -1)
        sum += num;
}while(num != -1);
printf("%d",sum);
```

# The use of do-wh

Notice proper indentation for while and do-while loops



The do-while loop is executed at least once

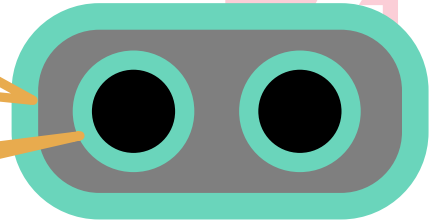
Recall: *read integers till you read the number -1and …*

```
int num, sum = 0;
scanf(“%d”, &num);
while(num != -1){
    sum += num;
    scanf(“%d”, &num);
}
printf("%d",sum);
```

```
int num, sum = 0;
do{
    scanf(“%d”, &num);
    if(num != -1)
        sum += num;
}while(num != -1);
printf("%d",sum);
```

# The use of do-wh

Notice proper indentation for while and do-while loops



The do-while loop is e

while and do-while equally powerful, sometimes one looks prettier, easier to read than the other

Recall: *read integers* *and …*

```
int num, sum = 0;
scanf(“%d”, &num);
while(num != -1){
    sum += num;
    scanf(“%d”, &num);
}
printf("%d",sum);
```

```
int num, sum = 0;
do{
    scanf(“%d”, &num);
    if(num != -1)
        sum += num;
}while(num != -1);
printf("%d",sum);
```

# More tips on using loops

# More tips on using loops



Take a positive integer n and print the pattern

# More tips on using loops



Take a positive integer n and print the pattern

```
1
1 2
1 2 3
1 2 3 4
...
1 2 3 4 ... n
```

# More tips on using loops



Take a positive integer n and print the pattern

Step 1: break up task into smaller tasks that are very similar and have to be repeated

```
1
1 2
1 2 3
1 2 3 4
...
1 2 3 4 ... n
```

# More tips on using loops



Take a positive integer n and print the pattern

Step 1: break up task into smaller tasks that are very similar and have to be repeated

May exist more than one way to do so – an art!

```
1
1 2
1 2 3
1 2 3 4
...
1 2 3 4 ... n
```

# More tips on using loops



Take a positive integer n and print the pattern

Step 1: break up task into smaller tasks that are very similar and have to be repeated

May exist more than one way to do so – an art!

```
1
1 2
1 2 3
1 2 3 4
...
1 2 3 4 ... n
```

# More tips on using loops

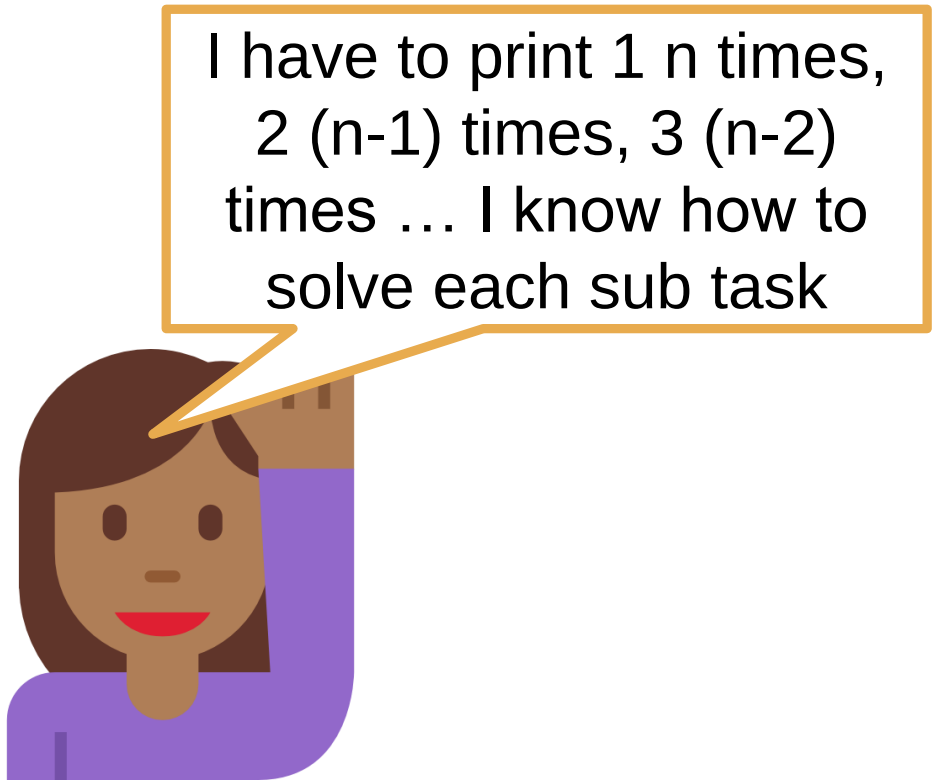


Take a positive integer n and print the pattern

```
1
1 2
1 2 3
1 2 3 4
...
1 2 3 4 ... n
```

Step 1: break up task into smaller tasks that are very similar and have to be repeated

May exist more than one way to do so – an art!



# More tips on using loops



1
1 2
1 2 3
1 2 3 4
…
1 2 3 4 … n

Take a positive integer n and print the pattern

Step 1: break up task into smaller tasks that are very similar and have to be repeated

May exist more than one way to do so – an art!

I have to print 1 n times, 2 (n-1) times, 3 (n-2) times … I know how to solve each sub task

```
for(j = 1; j <= n-i+1; j++)
    printf("%d\n", i);
```

# More tips on using loops

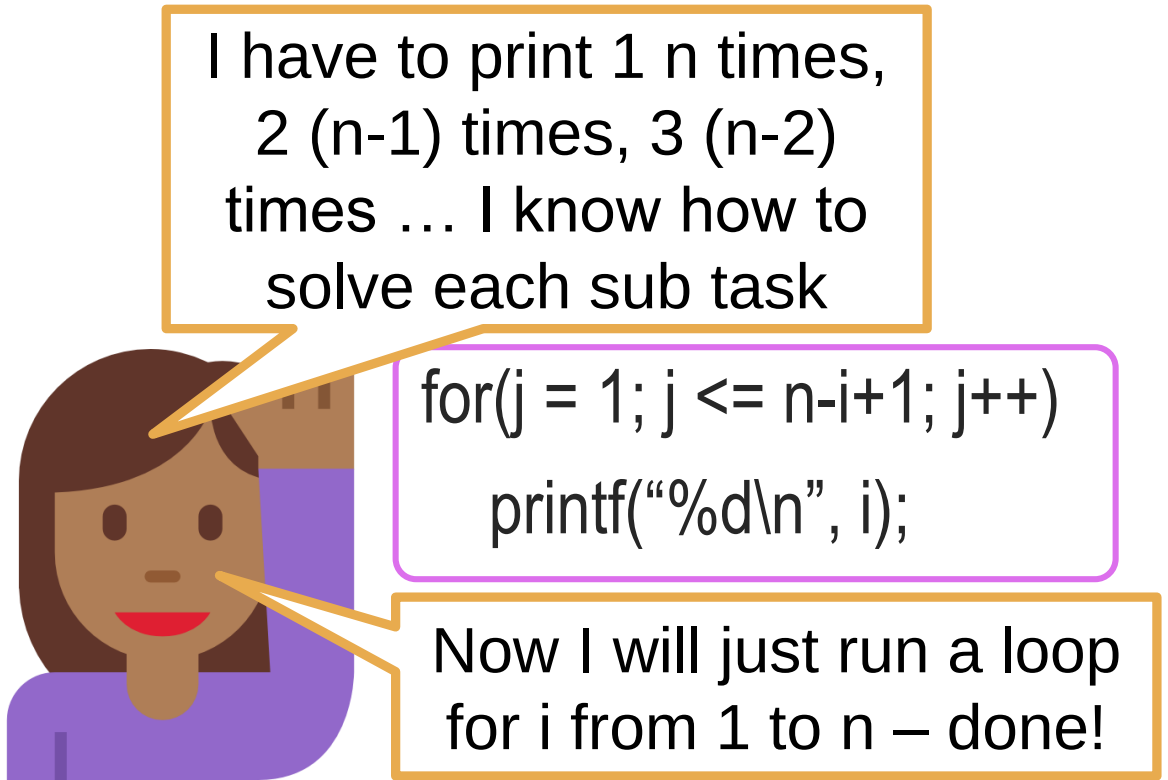


Take a positive integer n and print the pattern

```
1
1 2
1 2 3
1 2 3 4
...
1 2 3 4 ... n
```

Step 1: break up task into smaller tasks that are very similar and have to be repeated

May exist more than one way to do so – an art!



# More tips on using loops

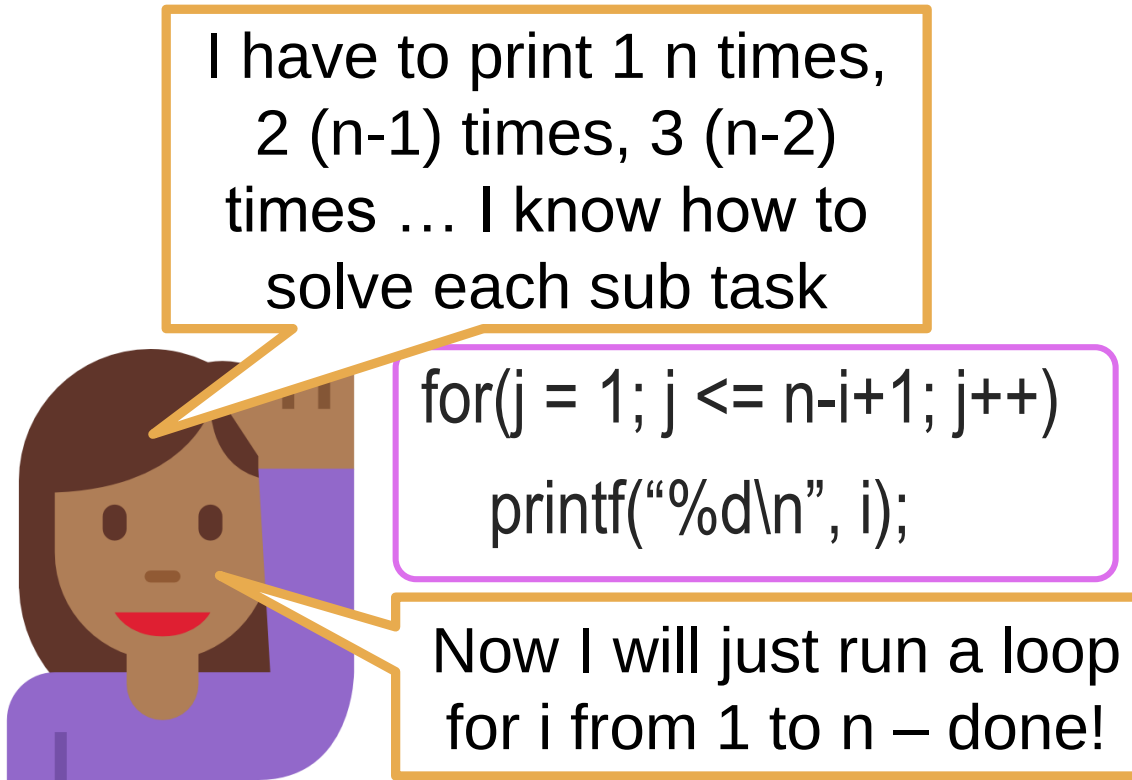


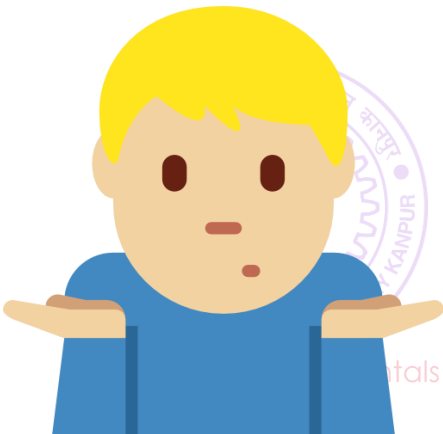
1
1 2
1 2 3
1 2 3 4
…
1 2 3 4 … n

Take a positive integer n and print the pattern

Step 1: break up task into smaller tasks that are very similar and have to be repeated

May exist more than one way to do so – an art!



# More tips on using loops

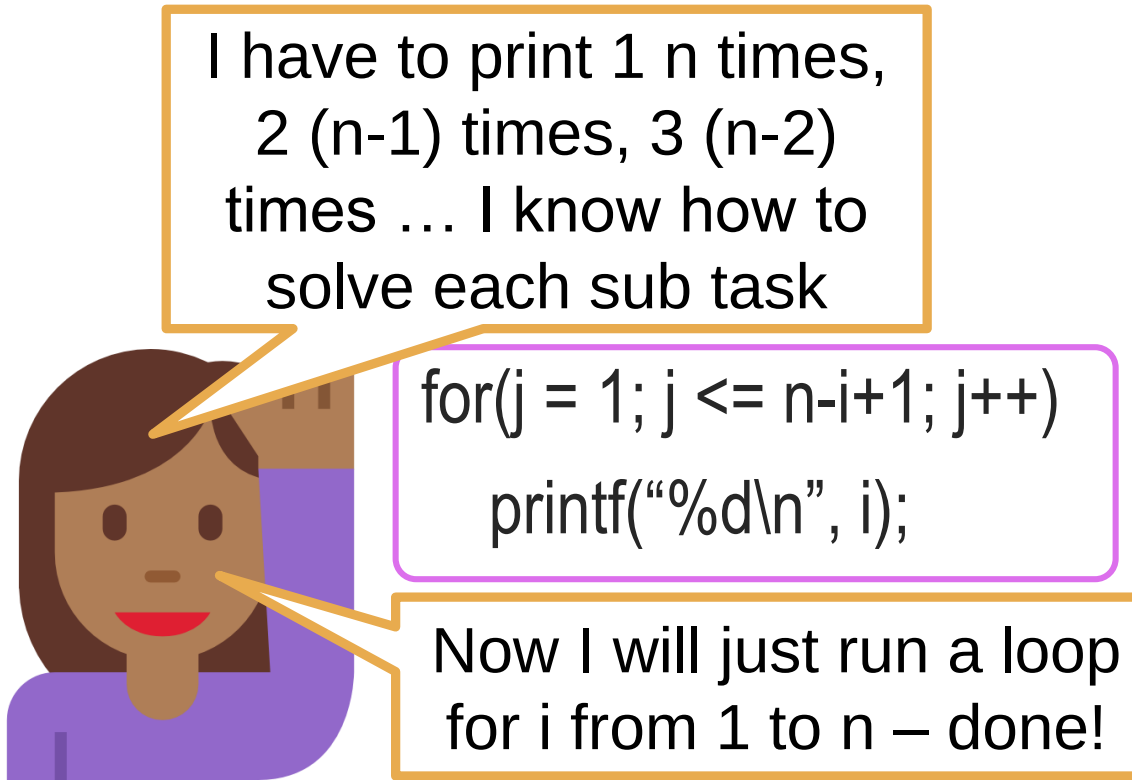


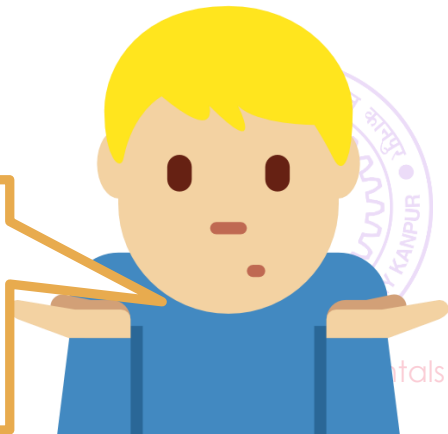
```
1
1 2
1 2 3
1 2 3 4
...
1 2 3 4 ... n
```

Take a positive integer n and print the pattern

Step 1: break up task into smaller tasks that are very similar and have to be repeated

May exist more than one way to do so – an art!



# More tips on using loops

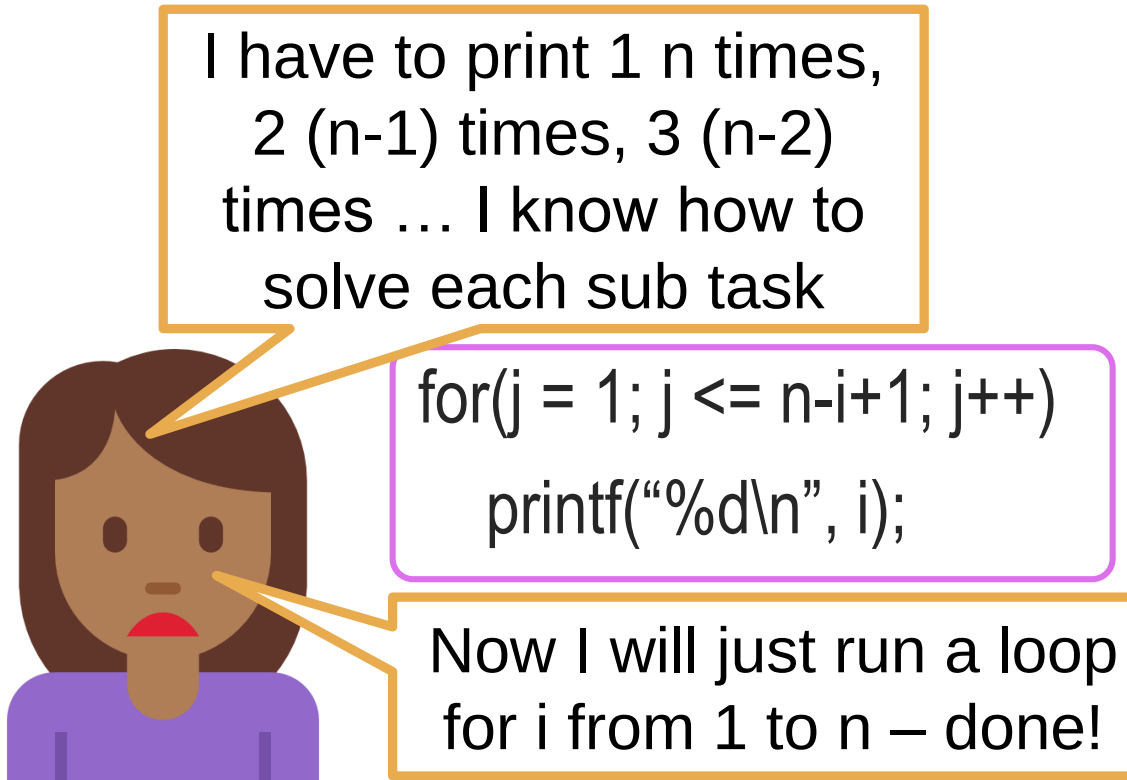


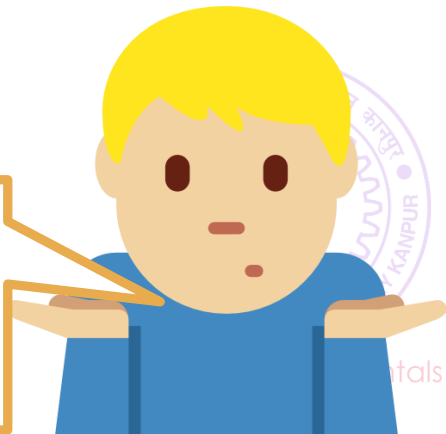
Take a positive integer n and print the pattern

```
1
1 2
1 2 3
1 2 3 4
...
1 2 3 4 ... n
```

Step 1: break up task into smaller tasks that are very similar and have to be repeated

May exist more than one way to do so – an art!



# More tips on using loops

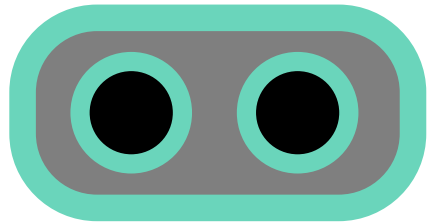


Take a positive integer n and print the pattern

```
1
1 2
1 2 3
1 2 3 4
...
1 2 3 4 ... n
```

Step 1: break up task into smaller tasks that are very similar and have to be repeated

May exist more than one way to do so – an art!

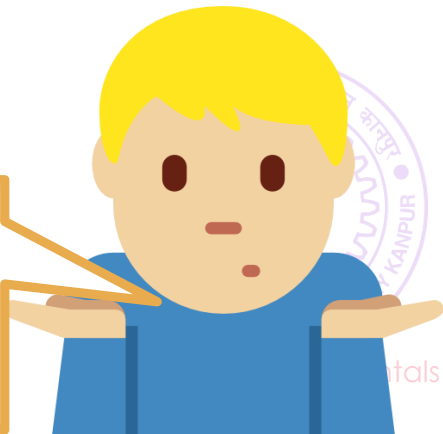


I have to print 1 n times, 2 (n-1) times, 3 (n-2) times … I know how to solve each sub task

```
for(j = 1; j <= n-i+1; j++)
    printf("%d\n", i);
```

Now I will just run a loop for i from 1 to n – done!

Once you have printed all the 1, how will you go back to previous line to start printing 2?



# More tips on using loops

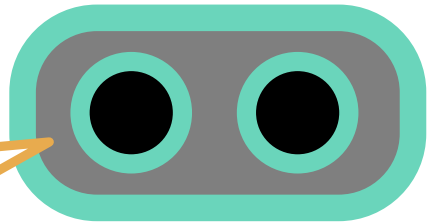


Take a positive integer n and print the pattern

```
1
1 2
1 2 3
1 2 3 4
...
1 2 3 4 ... n
```

Step 1: break up task into smaller tasks that are very similar and have to be repeated

May exist more than one way to do so – an art!



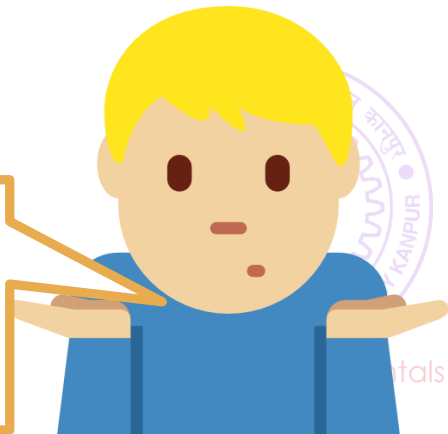
Better way to break up into smaller tasks: I have to print n lines. On line i, need to print digits 1 2 3 … i

I have to print 1 n times, 2 (n-1) times, 3 (n-2) times … I know how to solve each sub task

```
for(j = 1; j <= n-i+1; j++)
    printf("%d\n", i);
```

Now I will just run a loop for i from 1 to n – done!

Once you have printed all the 1, how will you go back to previous line to start printing 2?



# More tips on using loops

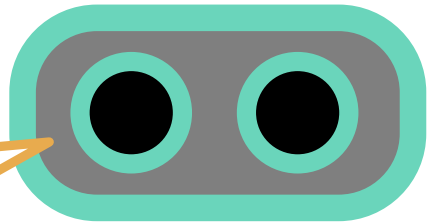


1
1 2
1 2 3
1 2 3 4
...
1 2 3 4 ... n

Take a positive integer n and print the pattern

Step 1: break up task into smaller tasks that are very similar and have to be repeated

May exist more than one way to do so – an art!



I have to print 1 n times, 2 (n-1) times, 3 (n-2) times … I know how to solve each sub task

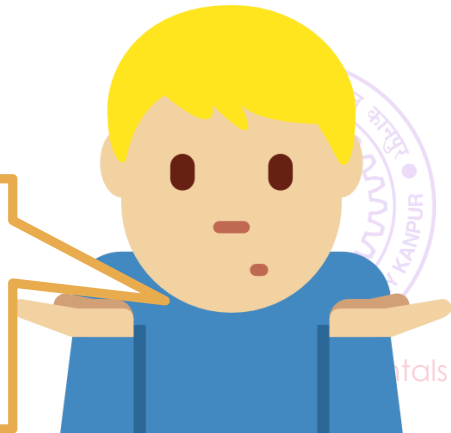
Better way to break up into smaller tasks: I have to print n lines. On line i, need to print digits 1 2 3 … i

```
for(j = 1; j <= n-i+1; j++)
    printf("%d\n", i);
```

```
for(j = 1; j <= i; j++)
    printf("%d ", j);
```

Now I will just run a loop for i from 1 to n – done!

Once you have printed all the 1, how will you go back to previous line to start printing 2?



# More tips on using loops

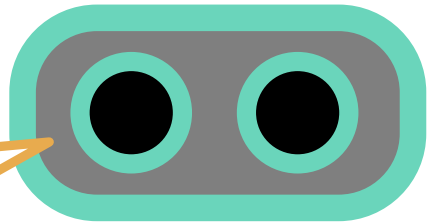


Take a positive integer n and print the pattern

```
1
1 2
1 2 3
1 2 3 4
...
1 2 3 4 ... n
```

Step 1: break up task into smaller tasks that are very similar and have to be repeated

May exist more than one way to do so – an art!



I have to print 1 n times, 2 (n-1) times, 3 (n-2) times … I know how to solve each sub task

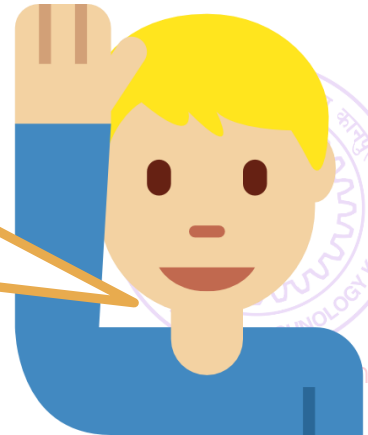
Better way to break up into smaller tasks: I have to print n lines. On line i, need to print digits 1 2 3 … i

```
for(j = 1; j <= n-i+1; j++)
    printf("%d\n", i);
```

```
for(j = 1; j <= i; j++)
    printf("%d ", j);
```

Now I will just run a loop for i from 1 to n – done!

Once you have printed all the 1, how will you go back to previous line to start printing 2?



# More tips on using loops

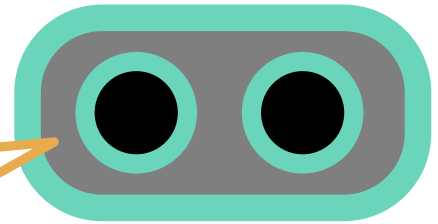


1
1 2
1 2 3
1 2 3 4
...
1 2 3 4 ... n

Take a positive integer n and print the pattern

Step 1: break up task into smaller tasks that are very similar and have to be repeated

May exist more than one way to do so – an art!



I have to print 1 n times, 2 (n-1) times, 3 (n-2) times … I know how to solve each sub task

Better way to break up into smaller tasks: I have to print n lines. On line i, need to print digits 1 2 3 … i

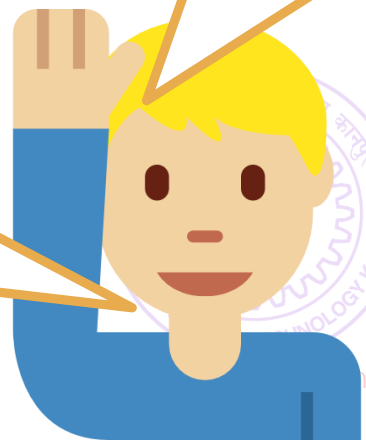
Take care of trailing spaces ☺

```
for(j = 1; j <= n-i+1; j++)
    printf("%d\n", i);
```

```
for(j = 1; j <= i; j++)
    printf("%d ", j);
```

Now I will just run a loop for i from 1 to n – done!

Once you have printed all the 1, how will you go back to previous line to start printing 2?



# More tips on using loops



1
1 2
1 2 3
1 2 3 4
…
1 2 3 4 … n

Take a positive integer n and print the pattern

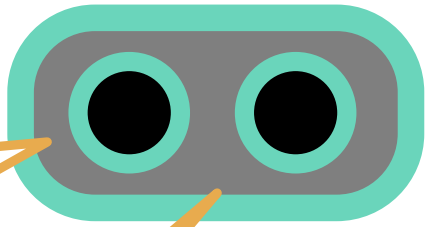
Step 1: break up task into smaller tasks that are very similar and have to be repeated

May exist more than one way to do so – an art!

I have to print 1 n times, 2 (n-1) times, 3 (n-2) times … I know how to solve each sub task

```
for(j = 1; j <= n-i+1; j++)
    printf("%d\n", i);
```

Now I will just run a loop for i from 1 to n – done!

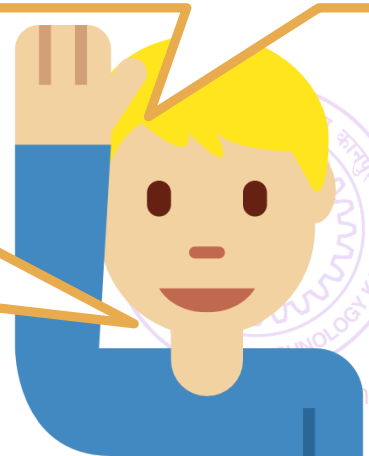


Better way to break up into smaller tasks: I have to print n lines. On line i, need to print digits 1 2 3 … i

```
for(j = 1; j <= i; j++)
    printf("%d ", j);
```

Very Good!

Take care of trailing spaces ☺



Once you have printed all the 1, how will you go back to previous line to start printing 2?

# Use of nested loops

# Use of nested loops



Don't be afraid of nested loops – they arise in tasks where the smaller tasks themselves require repetition ☺

# Use of nested loops



Don't be afraid of nested loops – they arise in tasks where the smaller tasks themselves require repetition ☺

Lets solve another problem – take a positive integer n and print the following pattern

# Use of nested loops



Don't be afraid of nested loops – they arise in tasks where the smaller tasks themselves require repetition ☺

Lets solve another problem – take a positive integer n and print the following pattern

1

2 3

3 4 5

4 5 6 7

…

n n+1 n+2 n+3 …

# Use of nested loops



Don't be afraid of nested loops – they arise in tasks where the smaller tasks themselves require repetition ☺

Lets solve another problem – take a positive integer n and print the following pattern

1

2 3

3 4 5

4 5 6 7

…

n n+1 n+2 n+3 …

# Use of nested loops



Don't be afraid of nested loops – they arise in tasks where the smaller tasks themselves require repetition ☺

Lets solve another problem – take a positive integer n and print the following pattern

1

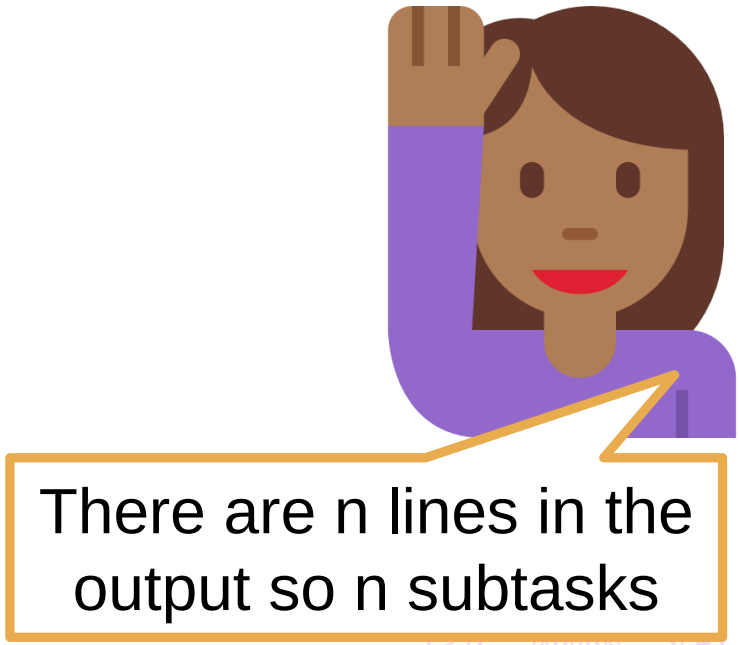
2 3

3 4 5

4 5 6 7

...

n n+1 n+2 n+3 ...



# Use of nested loops



Don't be afraid of nested loops – they arise in tasks where the smaller tasks themselves require repetition ☺

Lets solve another problem – take a positive integer n and print the following pattern

1

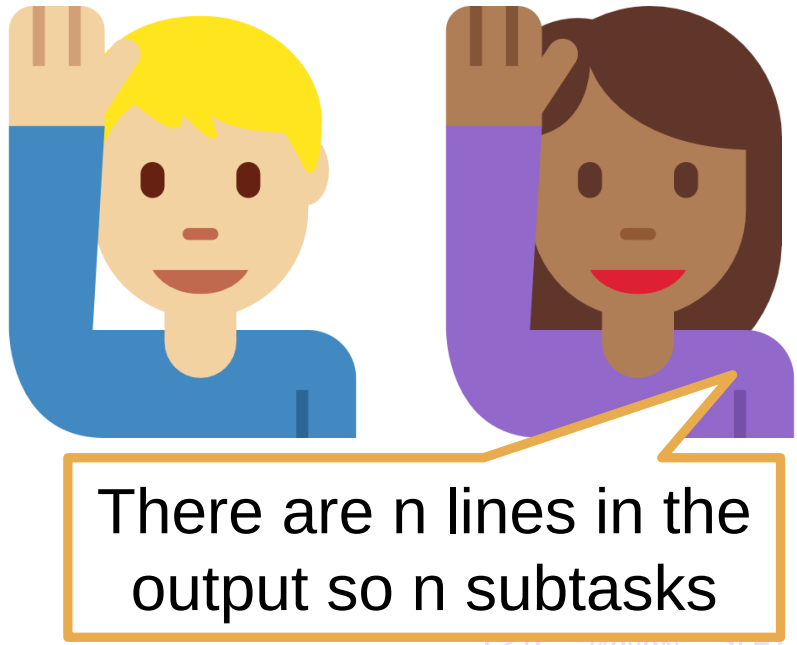
2 3

3 4 5

4 5 6 7

…

n n+1 n+2 n+3 …



# Use of nested loops



Don't be afraid of nested loops – they arise in tasks where the smaller tasks themselves require repetition ☺

Lets solve another problem – take a positive integer n and print the following pattern

1

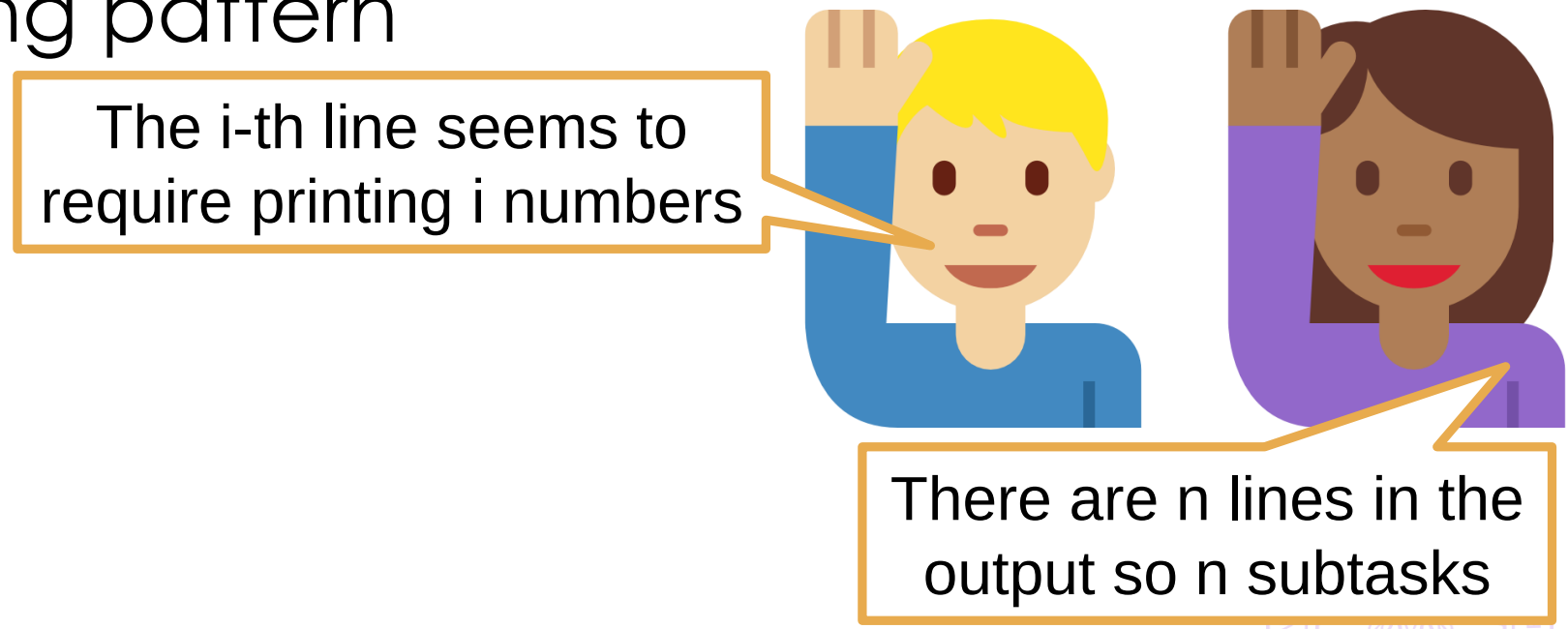
2 3

3 4 5

4 5 6 7

...

n n+1 n+2 n+3 ...



# Use of nested loops



Don't be afraid of nested loops – they arise in tasks where the smaller tasks themselves require repetition ☺

Lets solve another problem – take a positive integer n and print the following pattern

1

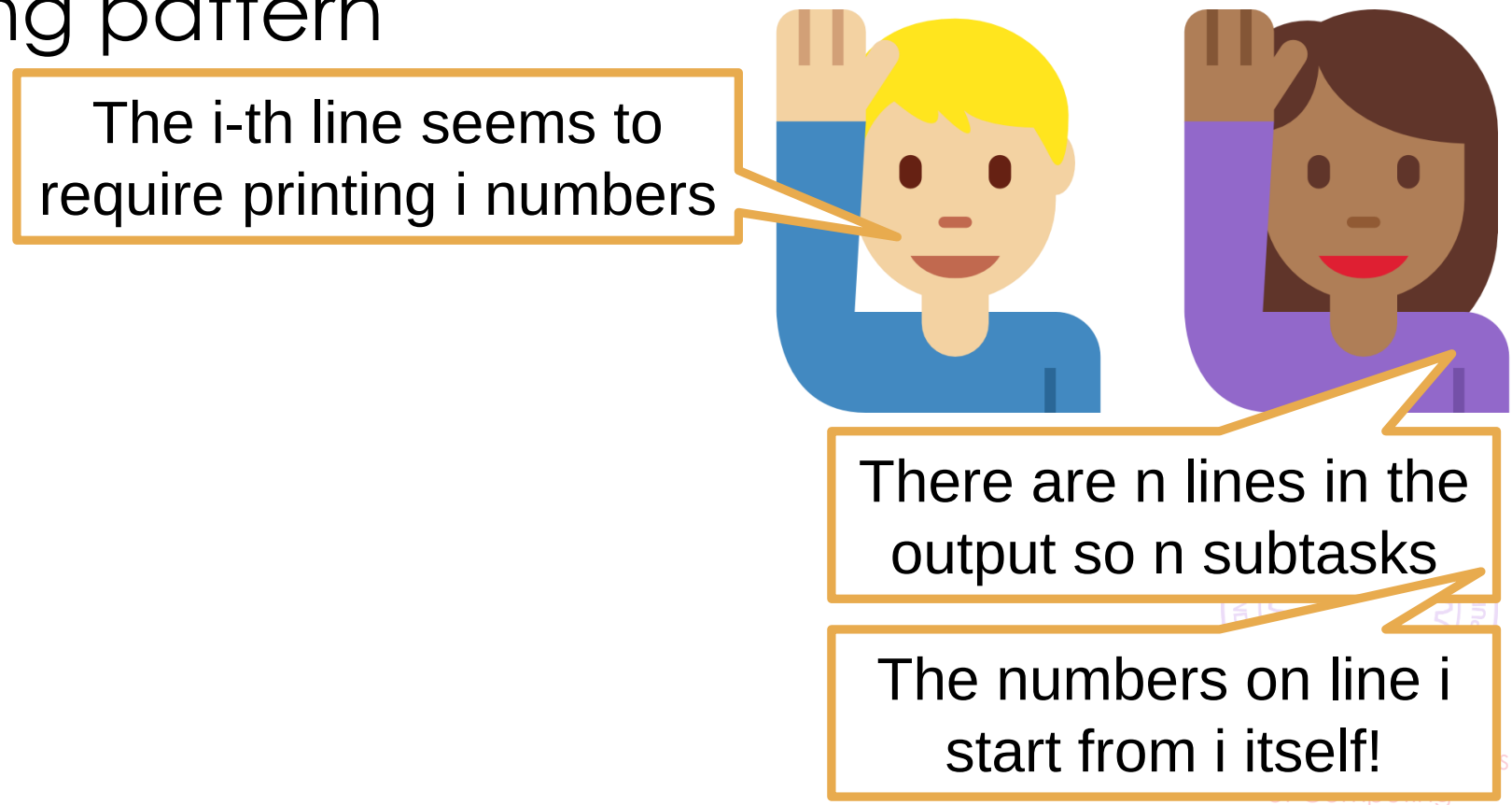
2 3

3 4 5

4 5 6 7

…

n n+1 n+2 n+3 …



# Use of nested loops



Don't be afraid of nested loops – they arise in tasks where the smaller tasks themselves require repetition ☺

Lets solve another problem – take a positive integer n and print the following pattern

1

2 3

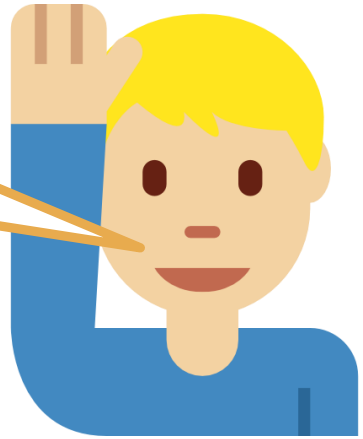
3 4 5

4 5 6 7

…

n n+1 n+2 n+3 …

The i-th line seems to require printing i numbers



```
for(j = i; j < i+i; j++)
    printf("%d ", j);
```

There are n lines in the output so n subtasks

The numbers on line i start from i itself!

# Use of nested loops



Don't be afraid of nested loops – they arise in tasks where the smaller tasks themselves require repetition ☺

Lets solve another problem – take a positive integer n and print the following pattern

1

2 3

3 4 5

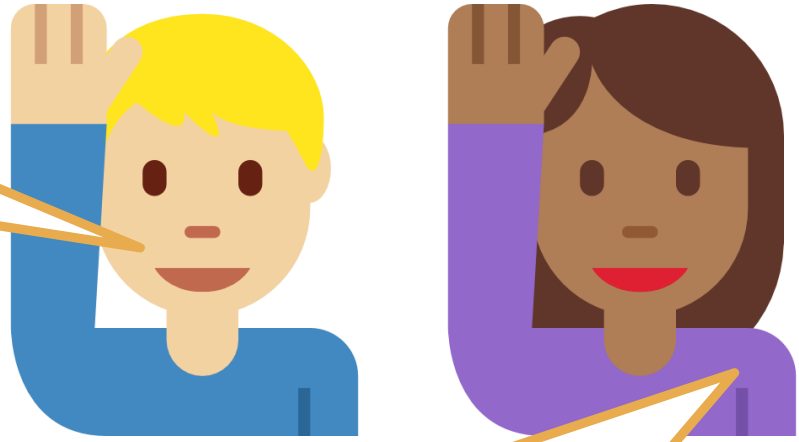
4 5 6 7

…

n n+1 n+2 n+3 …

The i-th line seems to require printing i numbers

```
for(i = 1; i <= n; i++){
    for(j = i; j < i+i; j++)
        printf("%d ", j);
    printf("\n");
}
```



There are n lines in the output so n subtasks

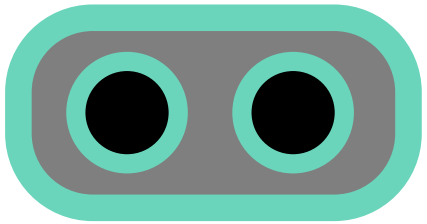
The numbers on line i start from i itself!

# Use of nested loops



Don't be afraid of nested loops – they arise in tasks where the smaller tasks themselves require repetition ☺

Lets solve another problem – take a positive integer n and print the following pattern



1

2 3

3 4 5

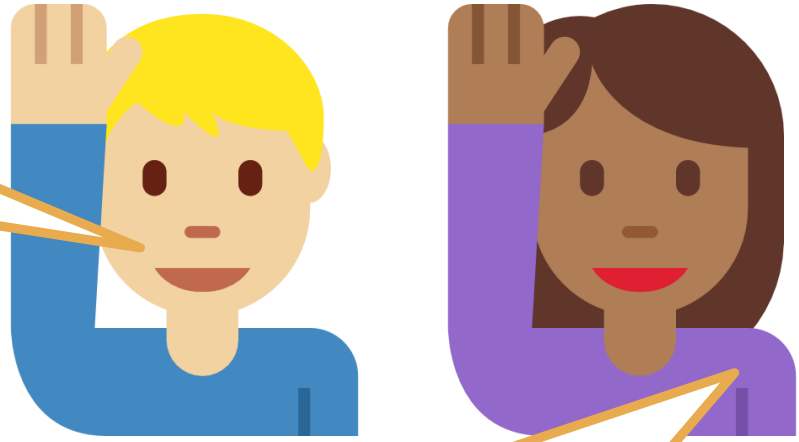
4 5 6 7

…

n n+1 n+2 n+3 …

The i-th line seems to require printing i numbers

```
for(i = 1; i <= n; i++){
    for(j = i; j < i+i; j++)
        printf("%d ", j);
    printf("\n");
}
```



There are n lines in the output so n subtasks

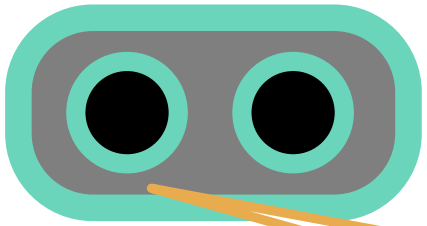
The numbers on line i start from i itself!

# Use of nested loops



Don't be afraid of nested loops – they arise in tasks where the smaller tasks themselves require repetition ☺

Lets solve another problem – take a positive integer n and print the following pattern



Excellent!

1

2 3

3 4 5

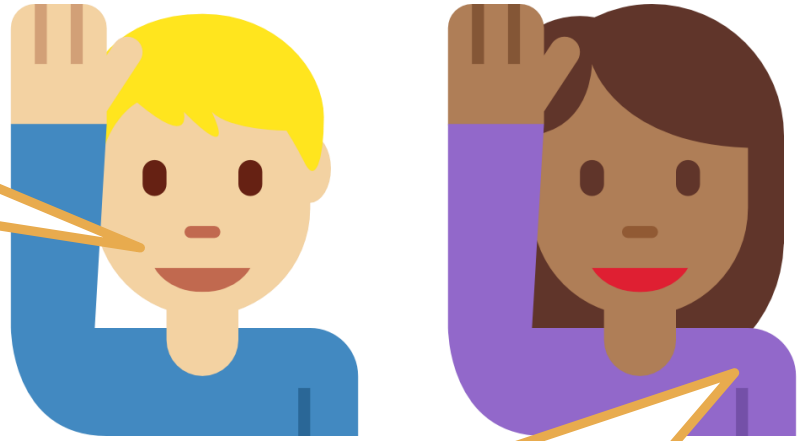
4 5 6 7

...

n n+1 n+2 n+3 ...

The i-th line seems to require printing i numbers

```
for(i = 1; i <= n; i++){
    for(j = i; j < i+i; j++)
        printf("%d ", j);
    printf("\n");
}
```



There are n lines in the output so n subtasks

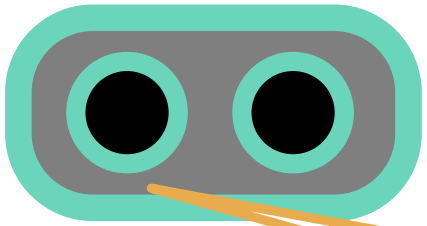
The numbers on line i start from i itself!

# Use of nested loops



Don't be afraid of nested loops – they arise in tasks where the smaller tasks themselves require repetition ☺

Lets solve another problem – take a positive integer n and print the following pattern



1

2 3

3 4 5

4 5 6 7

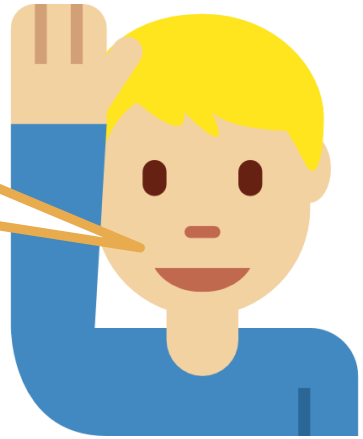
…

n n+1 n+2 n+3 …

Excellent!

Don't forget to take care of trailing spaces and trailing new lines

The i-th line seems to require printing i numbers



```
for(i = 1; i <= n; i++){
    for(j = i; j < i+i; j++)
        printf("%d ", j);
    printf("\n");
}
```

There are n lines in the output so n subtasks

The numbers on line i start from i itself!

# Use of nested loops

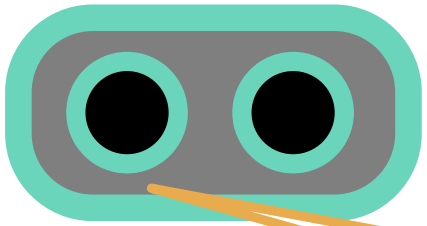


Don't be afraid of nested loops – they arise in tasks where the smaller tasks themselves require repetition ☺

Lets solve another problem – t itive integer n and print the following pattern

We sure will, Mr. C!

The i-th line seems to require printing i numbers

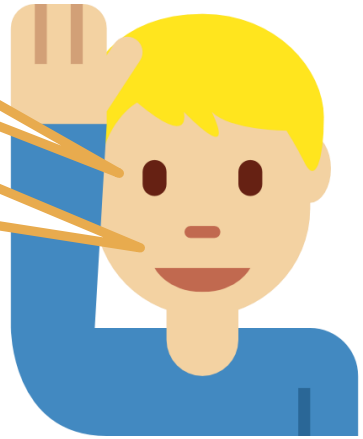


Excellent!

Don't forget to take care of trailing spaces and trailing new lines

```
1
2 3
3 4 5
4 5 6 7
...
n n+1 n+2 n+3 ...
```

```
for(i = 1; i <= n; i++){
    for(j = i; j < i+i; j++)
        printf("%d ", j);
    printf("\n");
}
```



There are n lines in the output so n subtasks

The numbers on line i start from i itself!

# Break




भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

# Break



Allows us to stop executing a loop and exit immediately

# Break



Allows us to stop executing a loop and exit immediately

- Even if stopping condition is still true

# Break



Allows us to stop executing a loop and exit immediately

- Even if stopping condition is still true
- Can be used inside a for loop, while loop, do-while loop

# Break



- Allows us to stop executing a loop and exit immediately
  - Even if stopping condition is still true
  - Can be used inside a for loop, while loop, do-while loop
- When to use break

# Break



- Allows us to stop executing a loop and exit immediately
  - Even if stopping condition is still true
  - Can be used inside a for loop, while loop, do-while loop
- When to use break
  - Avoid if possible

# Break



- Allows us to stop executing a loop and exit immediately
  - Even if stopping condition is still true
  - Can be used inside a for loop, while loop, do-while loop
- When to use break
  - Avoid if possible
  - Can make code error-prone and hard to read

# Break



- Allows us to stop executing a loop and exit immediately
  - Even if stopping condition is still true
  - Can be used inside a for loop, while loop, do-while loop
- When to use break
  - Avoid if possible
  - Can make code error-prone and hard to read
  - Used when one stopping condition not enough

# Break



## Allows us to stop executing a loop and exit immediately

- Even if stopping condition is still true
- Can be used inside a for loop, while loop, do-while loop

## When to use break

- Avoid if possible
- Can make code error-prone and hard to read
- Used when one stopping condition not enough
- Or to make code more elegant or just show-off!

# Break



## Allows us to stop executing a loop and exit immediately

- Even if stopping condition is still true
- Can be used inside a for loop, while loop, do-while loop

## When to use break

- Avoid if possible
- Can make code error-prone and hard to read
- Used when one stopping condition not enough
- Or to make code more elegant or just show-off!
- Allows us to avoid specifying a stopping condition

# Break



## Allows us to stop executing a loop and exit immediately

- Even if stopping condition is still true
- Can be used inside a for loop, while loop, do-while loop

## When to use break

- Avoid if possible
- Can make code error-prone and hard to read
- Used when one stopping condition not enough
- Or to make code more elegant or just show-off!
- Allows us to avoid specifying a stopping condition

```
int num, sum = 0;
while(1){
    scanf("%d", &num);
    if(num == -1) break;
    else sum += num;
}
printf("%d",sum);
```

# Break



Allows us to stop execu nd exit immediately

- Even if stopping condition is
- Can be used inside a for loop, while loop, hile loop

If we did not have break, infinite loop!

## When to use break

- Avoid if possible
- Can make code error-prone and hard to read
- Used when one stopping condition not enough
- Or to make code more elegant or just show-off!
- Allows us to avoid specifying a stopping condition

```
int num, sum = 0;
while(1){
    scanf("%d", &num);
    if(num == -1) break;
    else sum += num;
}
printf("%d",sum);
```

# Break



Allows us to stop execu nd exit immediately

- Even if stopping condition is
- Can be used inside a for loop, while loop, hile loop

If we did not have break, infinite loop!

## When to use break

- Avoid if possible
- Can make code error-prone and hard to read
- Used when one stopping condition not enough
- Or to make code more elegant or just show-off!
- Allows us to avoid specifying a stopping condition

```
int num, sum = 0;
while(1){
    scanf("%d", &num);
    if(num == -1) break;
    else sum += num;
}
printf("%d",sum);
```

# Break



Allows us to stop execu nd exit immediately

- Even if stopping condition is
- Can be used inside a for loop, while loop, hile loop

If we did not have break, infinite loop!

## When to use break

- Avoid if possible
- Can make code error-prone and hard to read
- Used when one stopping condition not enough
- Or to make code more elegant or just show-off!
- Allows us to avoid specifying a stopping condition

```
int num, sum = 0;
while(1){
    scanf("%d", &num);
    if(num == -1) break;
    else sum += num;
}
printf("%d",sum);
```

# Break



Allows us to stop execu[illegible]nd exit immediately

- Even if stopping condition is
- Can be used inside a for loop, while loop, [illegible]while loop

If we did not have break, infinite loop!

## When to use break

- Avoid if possible
- Can make code error-prone and hard to read
- Used when one stopping condition not enough
- Or to make code more elegant or just show-off!
- Allows us to avoid specifying a stopping condition

```
int num, sum = 0;
while(1){
    scanf("%d", &num);
    if(num == -1) break;
    else sum += num;
}
printf("%d",sum);
```

# Break



Allows us to stop execu nd exit immediately

- Even if stopping condition is
- Can be used inside a for loop, while loop, hile loop

If we did not have break, infinite loop!

## When to use break

- Avoid if possible
- Can make code error-prone and hard to read
- Used when one stopping condition not enough
- Or to make code more elegant or just show-off!
- Allows us to avoid specifying a stopping condition

```
int num, sum = 0;
while(1){
    scanf("%d", &num);
    if(num == -1) break;
    else sum += num;
}
printf("%d",sum);
```

# Break



Allows us to stop execu nd exit immediately

- Even if stopping condition is
- Can be used inside a for loop, while loop, hile loop

If we did not have break, infinite loop!

## When to use break

- Avoid if possible
- Can make code error-prone and hard to read
- Used when one stopping condition not enough
- Or to make code more elegant or just show-off!
- Allows us to avoid specifying a stopping condition

```
int num, sum = 0;
while(1){
    scanf("%d", &num);
    if(num == -1) break;
    sum += num;
}
printf("%d",sum);
```

# Break



Allows us to stop execu[illegible]nd exit immediately

- Even if stopping condition is
- Can be used inside a for loop, while loop, [illegible]while loop

If we did not have break, infinite loop!

## When to use break

- Avoid if possible
- Can make code error-prone and hard to read
- Used when one stopping condition not enough
- Or to make code more elegant or just show-off!
- Allows us to avoid specifying a stopping condition

Else condition not needed since Mr. C neglects all remaining statements in loop body upon encountering break;

```
int num, sum = 0;
while(1){
    scanf("%d", &num);
    if(num == -1) break;
    sum += num;
}
printf("%d",sum);
```

# Break



Allows us to stop execu nd exit immediately

- Even if stopping condition is
- Can be used inside a for loop, while loop, hile loop

If we did not have break, infinite loop!

## When to use break

- Avoid if possible
- Can make code error-prone and hard to read
- Used when one stopping condition not enough
- Or to make code more elegant or just show-off!
- Allows us to avoid specifying a stopping condition

Else condition not needed since Mr. C neglects all remaining statements in loop body upon encountering break;

```
int num, sum = 0;
while(1){
    scanf("%d", &num);
    if(num == -1) break;
    sum += num;
}
printf("%d",sum);
```

# Break



Allows us to stop execu nd exit immediately

- Even if stopping condition is
- Can be used inside a for loop, while loop, hile loop

If we did not have break, infinite loop!

## When to use break

- Avoid if possible
- Can make code error-prone and hard to read
- Used when one stopping condition not enough
- Or to make code more elegant or just show-off!
- Allows us to avoid specifying a stopping condition

Else condition not needed since Mr. C neglects all remaining statements in loop body upon encountering break;

```
int num, sum = 0;
while(1){
    scanf("%d", &num);
    if(num == -1) break;
    sum += num;
}
printf("%d",sum);
```

# Break



Allows us to stop execu[illegible]nd exit immediately

- Even if stopping condition is
- Can be used inside a for loop, while loop, [illegible]while loop

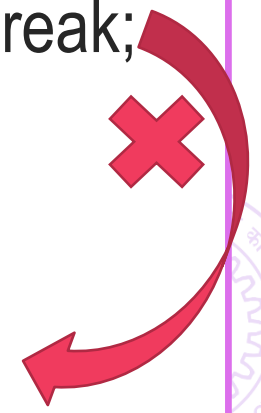
If we did not have break, infinite loop!

## When to use break

- Avoid if possible
- Can make code error-prone and hard to read
- Used when one stopping condition not enough
- Or to make code more elegant or just show-off!
- Allows us to avoid specifying a stopping condition

Else condition not needed since Mr. C neglects all remaining statements in loop body upon encountering break;

```
int num, sum = 0;
while(1){
    scanf("%d", &num);
    if(num == -1) break;
    sum += num;
}
printf("%d",sum);
```



# Fancy For using Break

# Fancy For using Break



```
for(init_expr; stop_expr; upd_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
```

# Fancy For using Break



```
for(init_expr; stop_expr; upd_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
```

```
init_expr;
for(;;){
    if(!(stop_expr)) break;
    statement1;
    statement2;
    ...
    upd_expr;
}
statement3;
statement4;
```

# For ... break

If we did not have break this would have been an infinite loop since no stop_expr

```
for(init_expr; stop_expr; upd_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
```

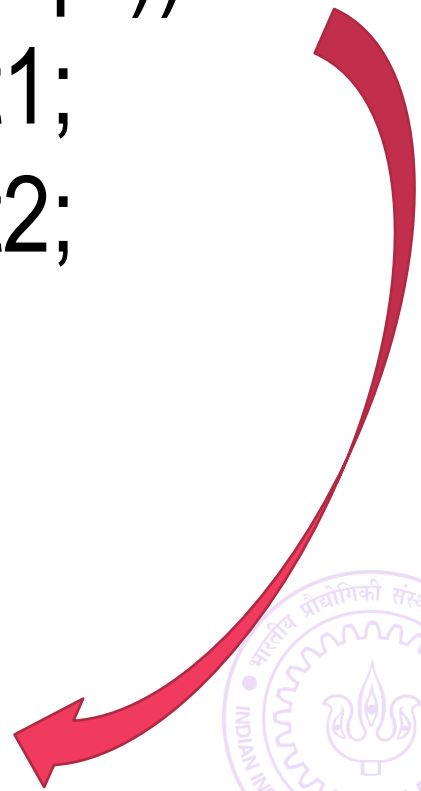
```
init_expr;
for(;;){
    if(!(stop_expr)) break;
    statement1;
    statement2;
    ...
    upd_expr;
}
statement3;
statement4;
```

# Fo ak

If we did not have break this would have been an infinite loop since no stop_expr

```
for(init_expr; stop_expr; upd_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
```

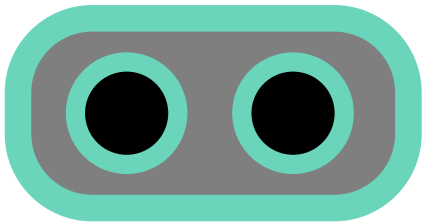
```
init_expr;
for(;;){
    if(!(stop_expr)) break;
    statement1;
    statement2;
    ...
    upd_expr;
}
statement3;
statement4;
```



# For ... break

If we did not have break this would have been an infinite loop since no stop_expr

```
for(init_expr; stop_expr; upd_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
```

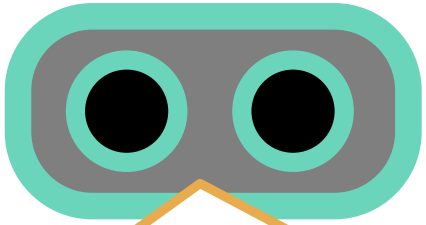


```
init_expr;
for(;;){
    if(!(stop_expr)) break;
    statement1;
    statement2;
    ...
    upd_expr;
}
statement3;
statement4;
```

# Fo ak

If we did not have break this would have been an infinite loop since no stop_expr

```
for(init_expr; stop_expr; upd_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
```



Remember, the order is stop_expr → body → update → stop_expr → …

```
init_expr;
for(;;){
    if(!(stop_expr)) break;
    statement1;
    statement2;
    ...
    upd_expr;
}
statement3;
statement4;
```

# Continue




भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

# Continue



Allows us to skip the rest of the statements in body of the loop

# Continue



Allows us to skip the rest of the statements in body of the loop

Upon encountering continue, Mr C thinks that body of loop is over

# Continue



Allows us to skip the rest of the statements in body of the loop

Upon encountering continue, Mr C thinks that body of loop is over

Loop not exited (unlike break)

# Continue



Allows us to skip the rest of the statements in body of the loop

Upon encountering continue, Mr C thinks that body of loop is over

Loop not exited (unlike break)

- If we say continue in for loop, update_expr evaluated, then stop condition checked

# Continue



Allows us to skip the rest of the statements in body of the loop

Upon encountering continue, Mr C thinks that body of loop is over

Loop not exited (unlike break)

- If we say continue in for loop, update_expr evaluated, then stop condition checked
- If we say continue in while or do-while loop, then stop condition checked

# Continue



Allows us to skip the rest of the statements in body of the loop

Upon encountering continue, Mr C thinks that body of loop is over

Loop not exited (unlike break)

- If we say continue in for loop, update_expr evaluated, then stop condition checked
- If we say continue in while or do-while loop, then stop condition checked
- In all cases, rest of body not executed

# Continue



Allows us to skip the rest of the statements in body of the loop

Upon encountering continue, Mr C thinks that body of loop is over

Loop not exited (unlike break)

- If we say continue in for loop, update_expr evaluated, then stop condition checked
- If we say continue in while or do-while loop, then stop condition checked
- In all cases, rest of body not executed

Read 100 integers and print sum of only positive numbers

# Continue



Allows us to skip the rest of the statements in body of the loop

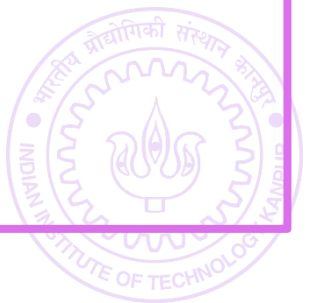
Upon encountering continue, Mr C thinks that body of loop is over

Loop not exited (unlike break)

- If we say continue in for loop, update_expr evaluated, then stop condition checked
- If we say continue in while or do-while loop, then stop condition checked
- In all cases, rest of body not executed

Read 100 integers and print sum of only positive numbers

```
int sum = 0, i, num;
for(i = 1; i <= 100; i++){
    scanf("%d", &num);
    if (num < 0){
        continue;
    }
    sum += num;
}
```



# Continue



Allows us to skip the rest of the statements in body of the loop

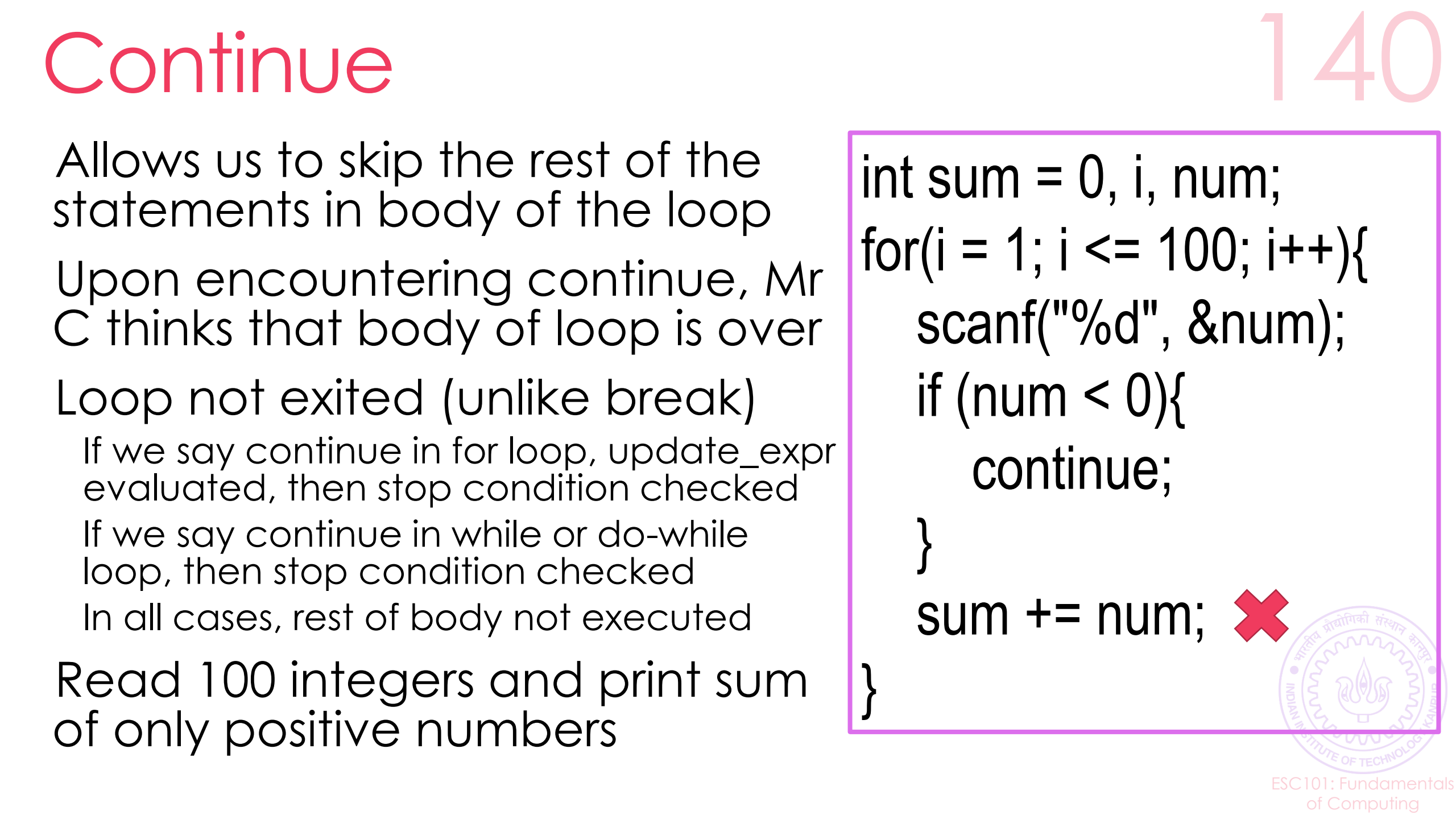
Upon encountering continue, Mr C thinks that body of loop is over

Loop not exited (unlike break)

- If we say continue in for loop, update_expr evaluated, then stop condition checked
- If we say continue in while or do-while loop, then stop condition checked
- In all cases, rest of body not executed

Read 100 integers and print sum of only positive numbers

```
int sum = 0, i, num;
for(i = 1; i <= 100; i++){
    scanf("%d", &num);
    if (num < 0){
        continue;
    }
    sum += num;
}
```



# Continue



Allows us to skip the rest of the statements in body of the loop

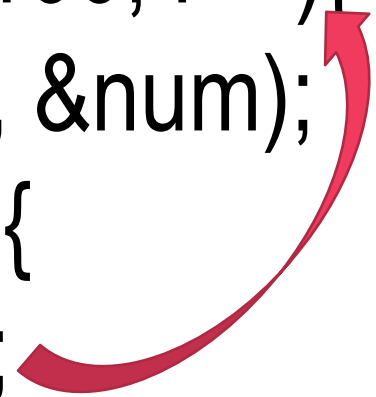
Upon encountering continue, Mr C thinks that body of loop is over

Loop not exited (unlike break)

- If we say continue in for loop, update_expr evaluated, then stop condition checked
- If we say continue in while or do-while loop, then stop condition checked
- In all cases, rest of body not executed

Read 100 integers and print sum of only positive numbers

```
int sum = 0, i, num;
for(i = 1; i <= 100; i++){
    scanf("%d", &num);
    if (num < 0){
        continue;
    }
    sum += num;
}
```



# Careful using break, continue

# Careful using break, continue



If there are nested loops, a break inside the inner loop will apply only to the inner loop

# Careful using break, continue



If there are nested loops, a break inside the inner loop will apply only to the inner loop

```
for (i = 0; i < 100; i++){
    for (j = 0; j < 100; j++){
        if (…) break;
    }
    statement1;
}
statement2
```

# Careful using break, continue



If there are nested loops, a break inside the inner loop will apply only to the inner loop

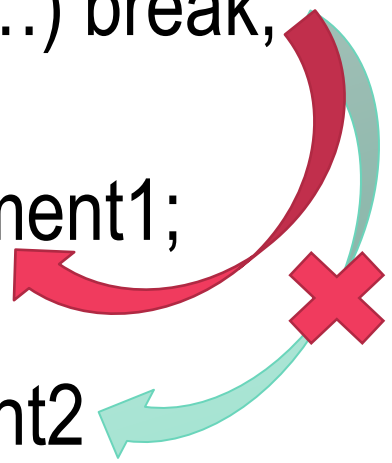
```
for (i = 0; i < 100; i++){
    for (j = 0; j < 100; j++){
        if (…) break;
    }
    statement1;
}
statement2
```

# Careful using break, continue



If there are nested loops, a break inside the inner loop will apply only to the inner loop

```
for (i = 0; i < 100; i++){
    for (j = 0; j < 100; j++){
        if (…) break;
    }
    statement1;
}
statement2
```

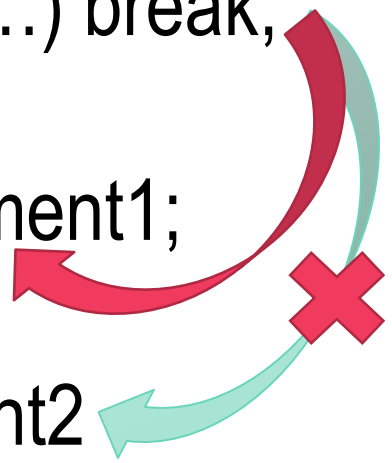


# Careful using break, continue



If there are nested loops, a break inside the inner loop will apply only to the inner loop, same with continue

```
for (i = 0; i < 100; i++){
    for (j = 0; j < 100; j++){
        if (…) break;
    }
    statement1;
}
statement2
```



# Careful using break, continue



If there are nested loops, a break inside the inner loop will apply only to the inner loop, same with continue

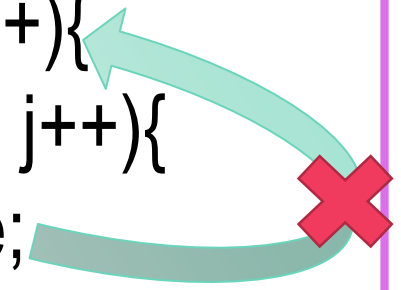
```
for (i = 0; i < 100; i++){
    for (j = 0; j < 100; j++){
        if (…) continue;
    }
    statement1;
}
statement2
```

# Careful using break, continue



If there are nested loops, a break inside the inner loop will apply only to the inner loop, same with continue

```
for (i = 0; i < 100; i++){
    for (j = 0; j < 100; j++){
        if (…) continue;
    }
    statement1;
}
statement2
```

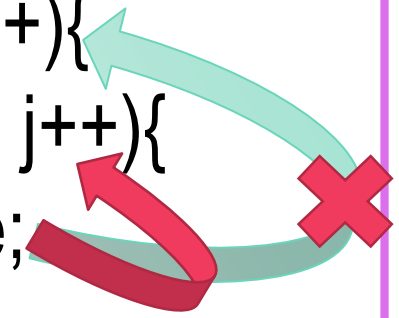


# Careful using break, continue



If there are nested loops, a break inside the inner loop will apply only to the inner loop, same with continue

```
for (i = 0; i < 100; i++){
    for (j = 0; j < 100; j++){
        if (…) continue;
    }
    statement1;
}
statement2
```



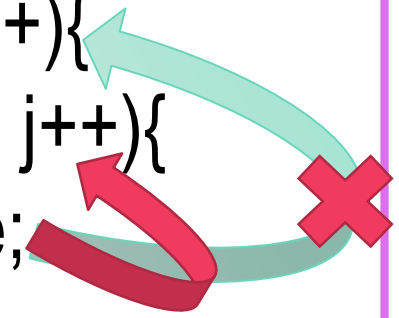
# Careful using break, continue



If there are nested loops, a break inside the inner loop will apply only to the inner loop, same with continue

Be careful not to create an infinite loop using continue if you bypass any update steps.

```
for (i = 0; i < 100; i++){
    for (j = 0; j < 100; j++){
        if (…) continue;
    }
    statement1;
}
statement2
```



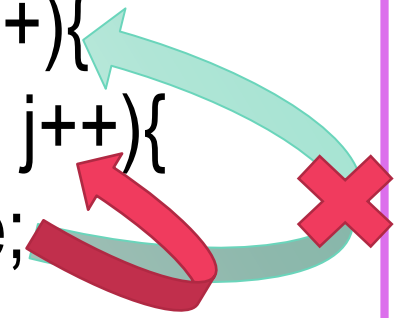
# Careful using break, continue



If there are nested loops, a break inside the inner loop will apply only to the inner loop, same with continue

Be careful not to create an infinite loop using continue if you bypass any update steps.

```
for (i = 0; i < 100; i++){
    for (j = 0; j < 100; j++){
        if (…) continue;
    }
    statement1;
}
statement2
```



```
int i = 0, sum = 0, num;
while (i < 100){
    scanf("%d", &num);
    if (num < 0) continue;
    sum += num;
    i++;
}
```

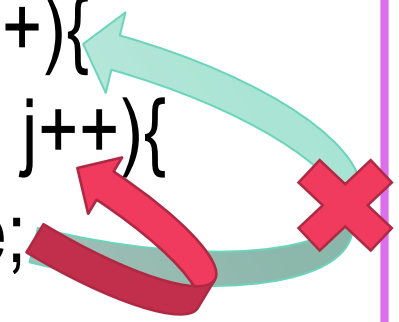
# Careful using break, continue



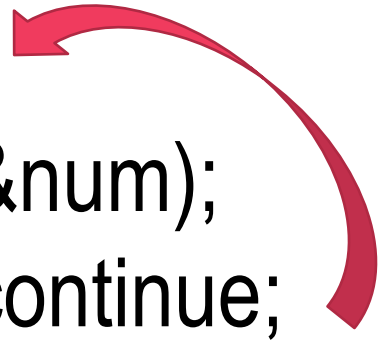
If there are nested loops, a break inside the inner loop will apply only to the inner loop, same with continue

Be careful not to create an infinite loop using continue if you bypass any update steps.

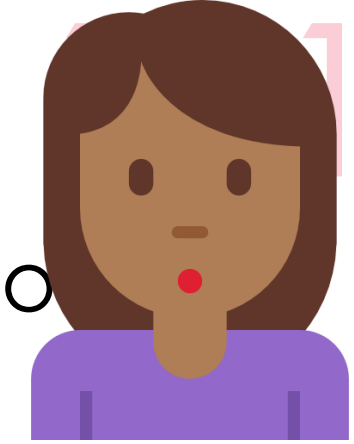
```
for (i = 0; i < 100; i++){
    for (j = 0; j < 100; j++){
        if (…) continue;
    }
    statement1;
}
statement2
```



```
int i = 0, sum = 0, num;
while (i < 100){
    scanf("%d", &num);
    if (num < 0) continue;
    sum += num;
    i++;
}
```



# Careful using break, continue



If there are nested loops, a break inside the inner lo
apply only to the inner loop, same with continue

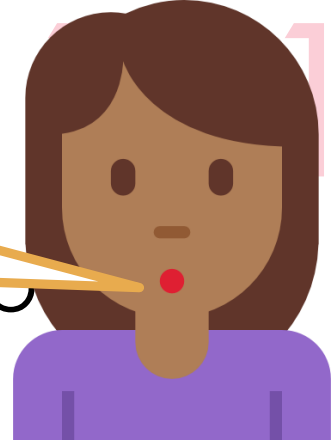
Be careful not to create an infinite loop using continue if you bypass any update steps.

```
for (i = 0; i < 100; i++){
    for (j = 0; j < 100; j++){
        if (…) continue;
    }
    statement1;
}
statement2
```

```
int i = 0, sum = 0, num;
while (i < 100){
    scanf("%d", &num);
    if (num < 0) continue;
    sum += num;
    i++;
}
```

# Careful u e

If we have a sequence of 100000 negative numbers, it will read all of them even though we wanted to read only first 100 numbers – continue statement skipping counter update



If there are ne ner lo apply only to the inner loop, same with continue

Be careful not to create an infinite loop using continue if you bypass any update steps.

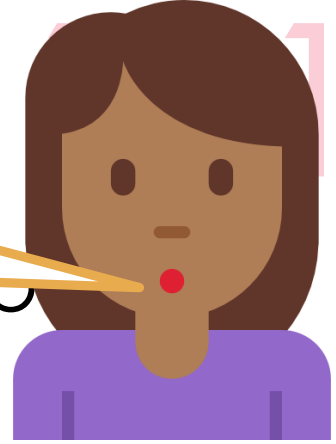
```
for (i = 0; i < 100; i++){
    for (j = 0; j < 100; j++){
        if (…) continue;
    }
    statement1;
}
statement2
```

```
int i = 0, sum = 0, num;
while (i < 100){
    scanf("%d", &num);
    if (num < 0) continue;
    sum += num;
    i++;
}
```

# Careful u e

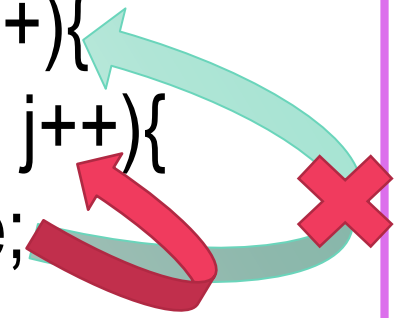
If there are ne inner lo apply only to the inner loop, same with continue

If we have a sequence of 100000 negative numbers, it will read all of them even though we wanted to read only first 100 numbers – continue statement skipping counter update



Be careful not to create an infinite loop using continue if you bypass any update steps.

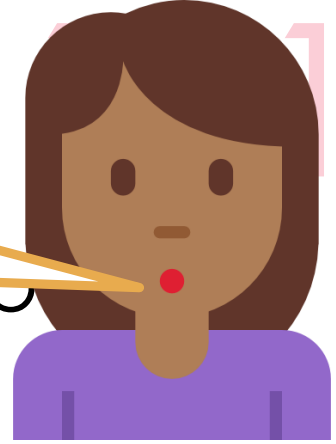
```
for (i = 0; i < 100; i++){
    for (j = 0; j < 100; j++){
        if (…) continue;
    }
    statement1;
}
statement2
```



```
int i = 0, sum = 0, num;
while (i < 100){
    i++;
    scanf("%d", &num);
    if (num < 0) continue;
    sum += num;
}
```

# Careful u e

If we have a sequence of 100000 negative numbers, it will read all of them even though we wanted to read only first 100 numbers – continue statement skipping counter update



If there are ne inner lo apply only to the inner loop, same with continue

Be careful not to create an infinite loop using continue if you bypass any update steps.

```
for (i = 0; i < 100; i++){
    for (j = 0; j < 100; j++){
        if (…) continue;
    }
    statement1;
}
statement2
```

```
int i = 0, sum = 0, num;
while (i < 100){
    i++;
    scanf("%d", &num);
    if (num < 0) continue;
    sum += num;
}
```

# Careful using break, continue

If there are nested loops, a break inside the inner loop will apply only to the inner loop, same with continue

Be careful not to create an infinite loop using continue if you bypass any update steps.

```
for (i = 0; i < 100; i++){
    for (j = 0; j < 100; j++){
        if (…) continue;
    }
    statement1;
}
statement2
```

```
int i = 0, sum = 0, num;
while (i < 100){
    i++;
    scanf("%d", &num);
    if (num < 0) continue;
    sum += num;
}
```

# Careful using bre e

If there are nested loops, c inner apply only to the inner loop, same with continue

Much better, always updates counter whether skipping the sum+=num step using continue or not

Be careful not to create an infinite loop using continue if you bypass any update steps.

```
for (i = 0; i < 100; i++){
    for (j = 0; j < 100; j++){
        if (…) continue;
    }
    statement1;
}
statement2
```

```
int i = 0, sum = 0, num;
while (i < 100){
    i++;
    scanf("%d", &num);
    if (num < 0) continue;
    sum += num;
}
```

# Careful using break, continue

# Careful using break, continue



Excessive use of break and continue can make your program error-prone, and hard for you to correct

# Careful using break, continue



Excessive use of break and continue can make your program error-prone, and hard for you to correct

If you have 10 break statements inside the same loop body, you will have a hard time figuring out which one caused your loop to end ☹

# Careful using break, continue



Excessive use of break and continue can make your program error-prone, and hard for you to correct

If you have 10 break statements inside the same loop body, you will have a hard time figuring out which one caused your loop to end ☹

If you have 10 continue statements inside the same loop body, you will have a hard time figuring out why body statements are not getting executed.

# Careful using break, continue



Excessive use of break and continue can make your program error-prone, and hard for you to correct

If you have 10 break statements inside the same loop body, you will have a hard time figuring out which one caused your loop to end ☹

If you have 10 continue statements inside the same loop body, you will have a hard time figuring out why body statements are not getting executed.

Should not misuse break, continue - used in moderation these can result in nice, beautiful code

# Careful using break, continue



Excessive use of break and continue can make your program error-prone, and hard for you to correct

If you have 10 break statements inside the same loop body, you will have a hard time figuring out which one caused your loop to end ☹

If you have 10 continue statements inside the same loop body, you will have a hard time figuring out why body statements are not getting executed.

Should not misuse break, continue - used in moderation these can result in nice, beautiful code

Tomorrow: more elegant alternatives to break, continue