

COMP
110

While Loops

First, Review

Conditionals:

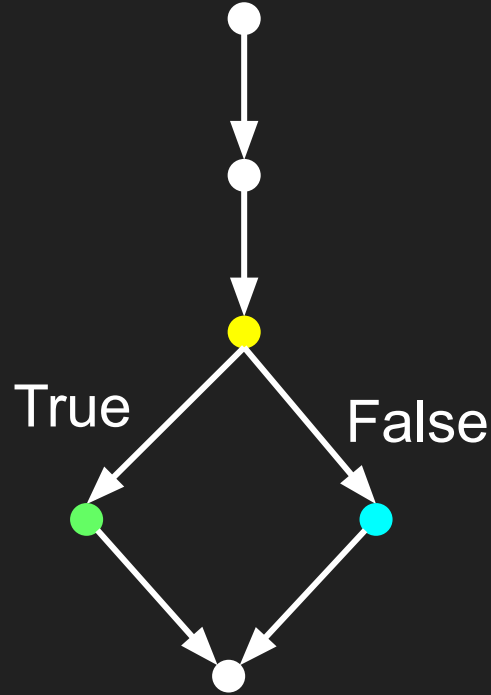
if <something>:

<do something>

else:

<do something else>

<continue program>



First, Review

Conditionals:

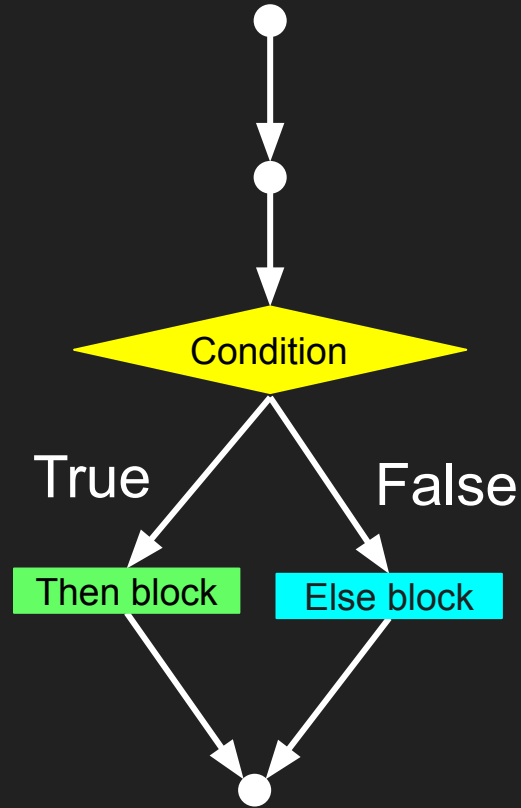
if <something>:

<do something>

else:

<do something else>

<continue program>



First, Review

Conditionals:

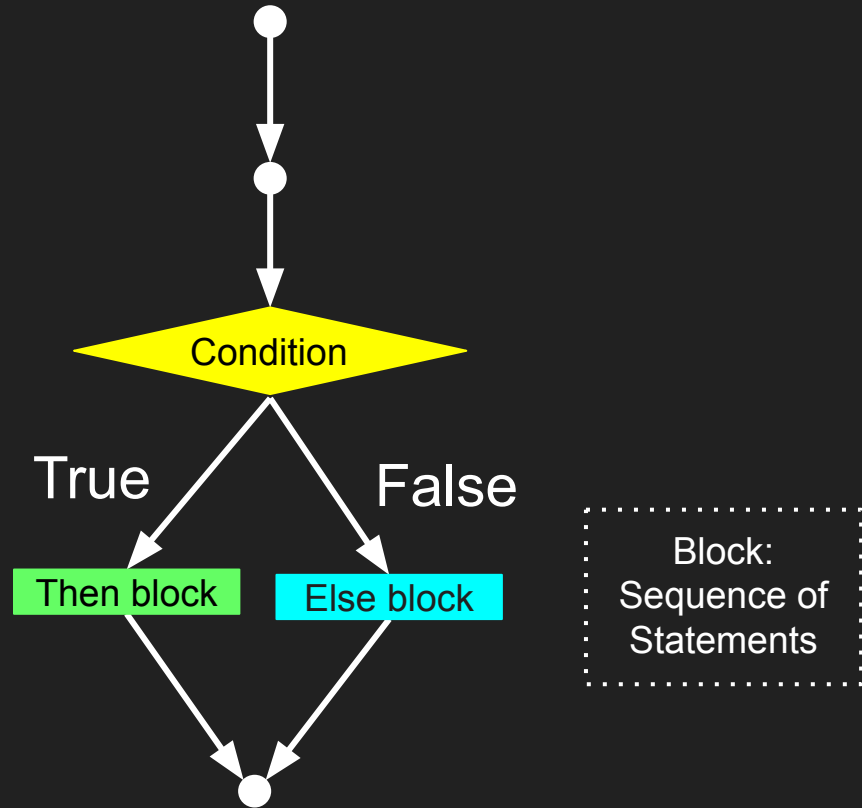
if <something>:

<do something>

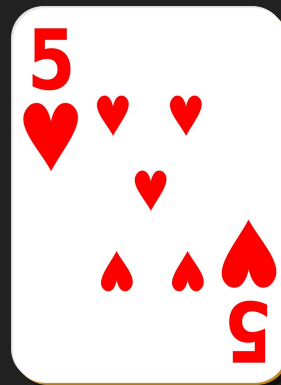
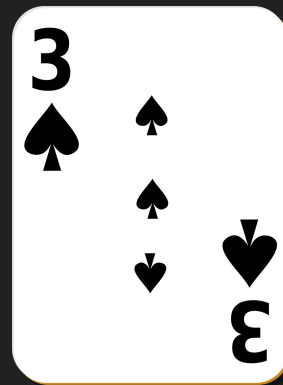
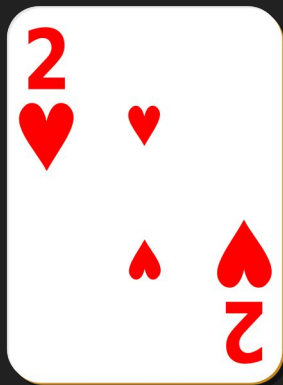
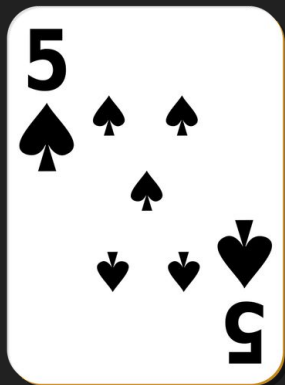
else:

<do something else>

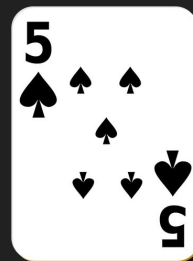
<continue program>



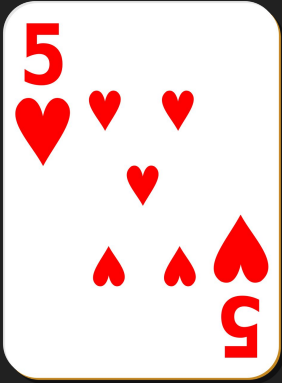
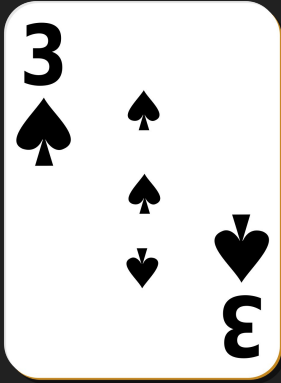
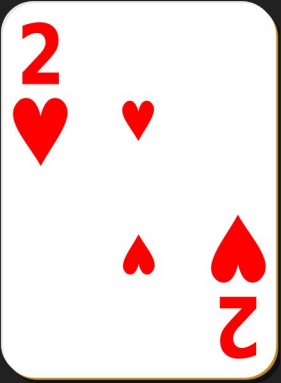
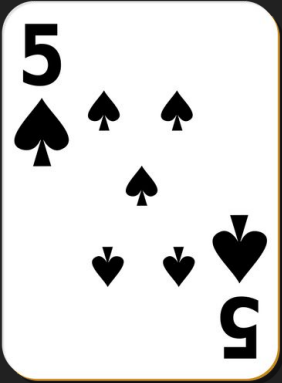
Finding the Lowest Card



Low card:



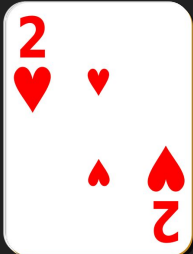
Finding the Lowest Card



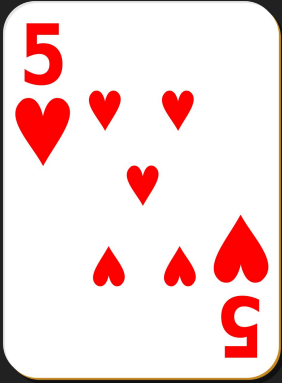
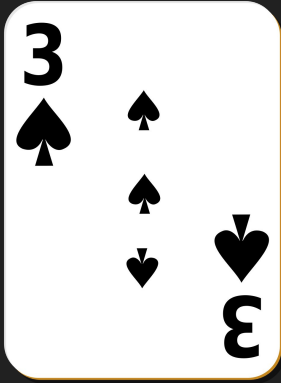
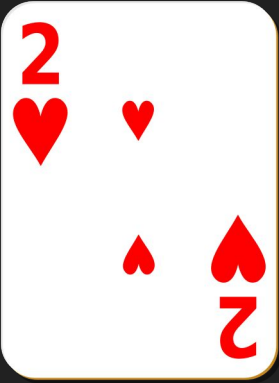
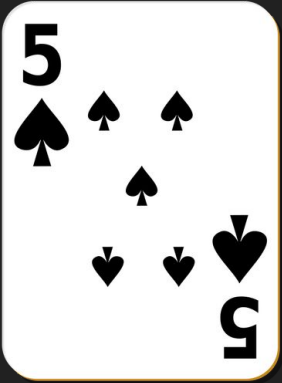
2 < 5?



Low card:

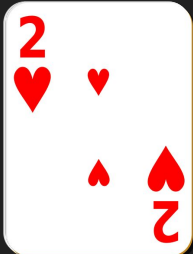


Finding the Lowest Card

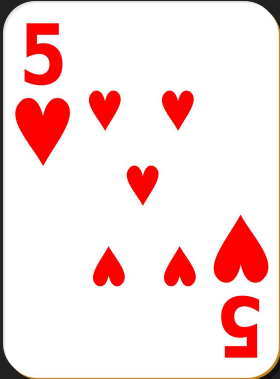
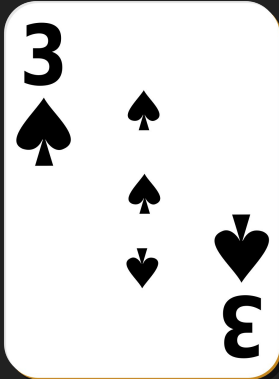
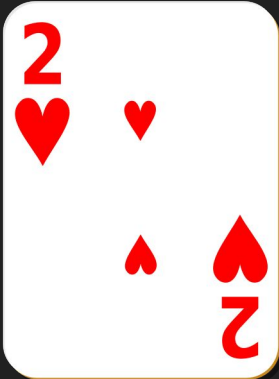
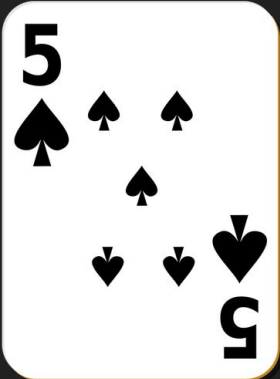


3 < 2? 

Low card:

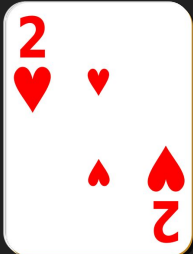


Finding the Lowest Card



5 < 2? 

Low card:



Finding the Lowest Card

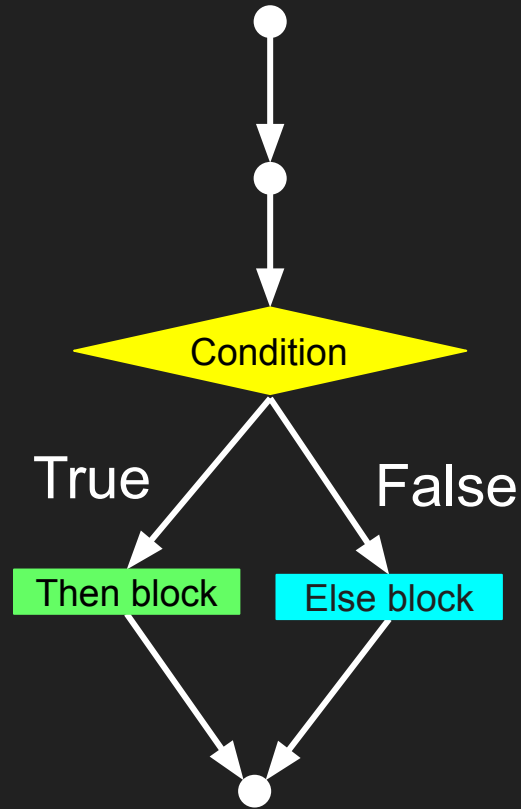
Finding the low card pseudocode:

1 lowest_card = first card in deck

2 Repeatedly until end of deck:

3 if current_card < lowest_card:

4 lowest_card = current_card



Finding the Lowest Card

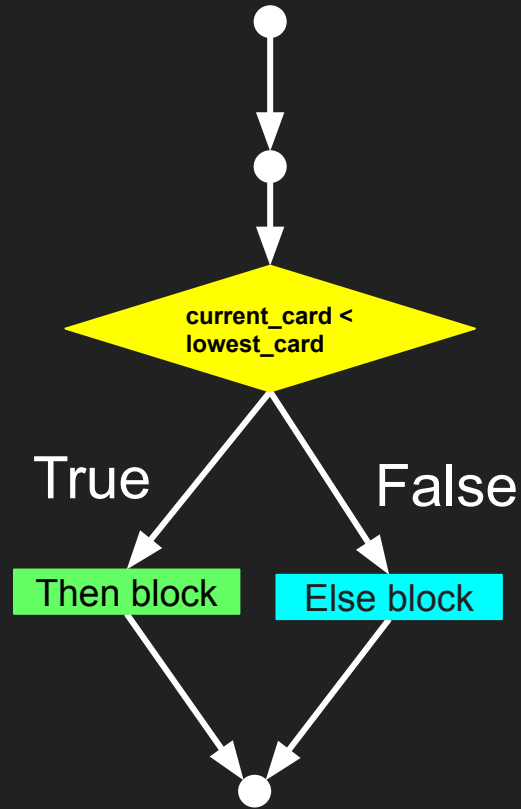
Finding the low card pseudocode:

1 lowest_card = first card in deck

2 Repeatedly until end of deck:

3 if current_card < lowest_card:

4 lowest_card = current_card



Finding the Lowest Card

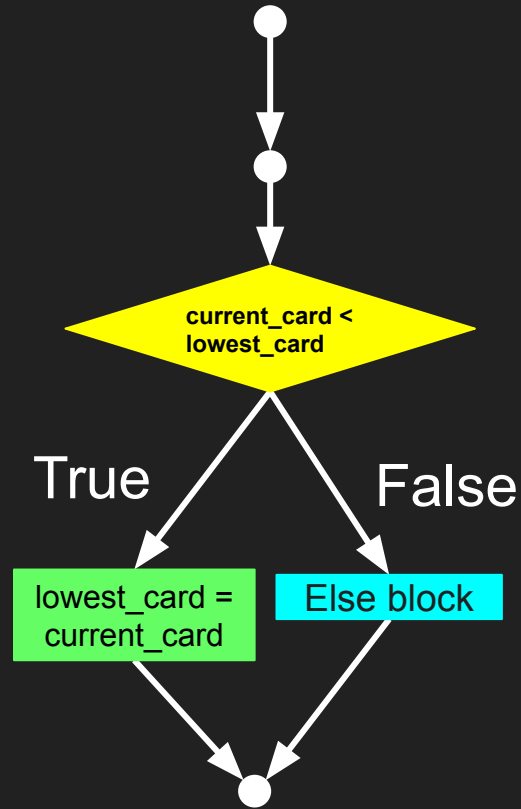
Finding the low card pseudocode:

1 lowest_card = first card in deck

2 Repeatedly until end of deck:

3 if current_card < lowest_card:

4 lowest_card = current_card



Finding the Lowest Card

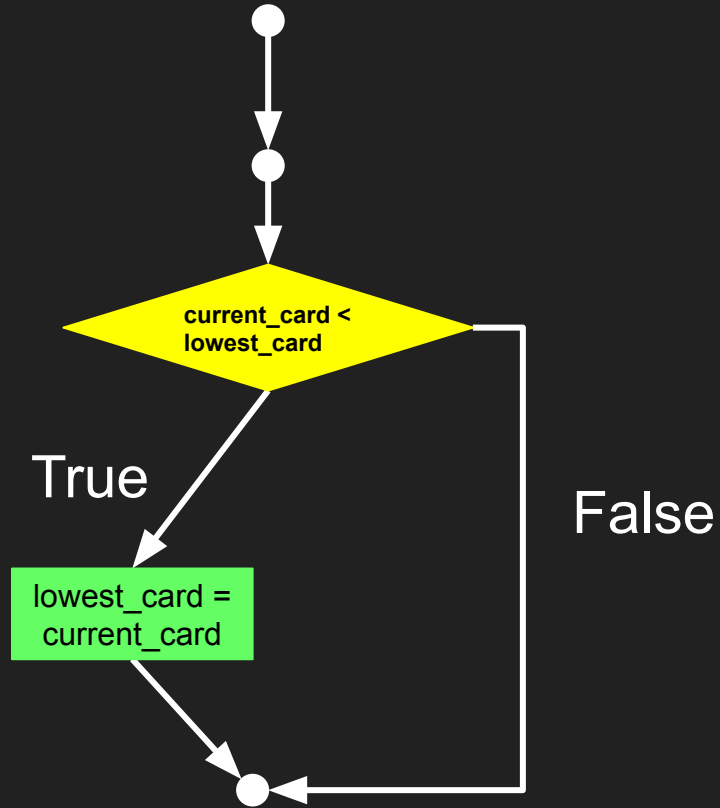
Finding the low card pseudocode:

1 lowest_card = first card in deck

2 Repeatedly until end of deck:

3 if current_card < lowest_card:

4 lowest_card = current_card



Finding the Lowest Card

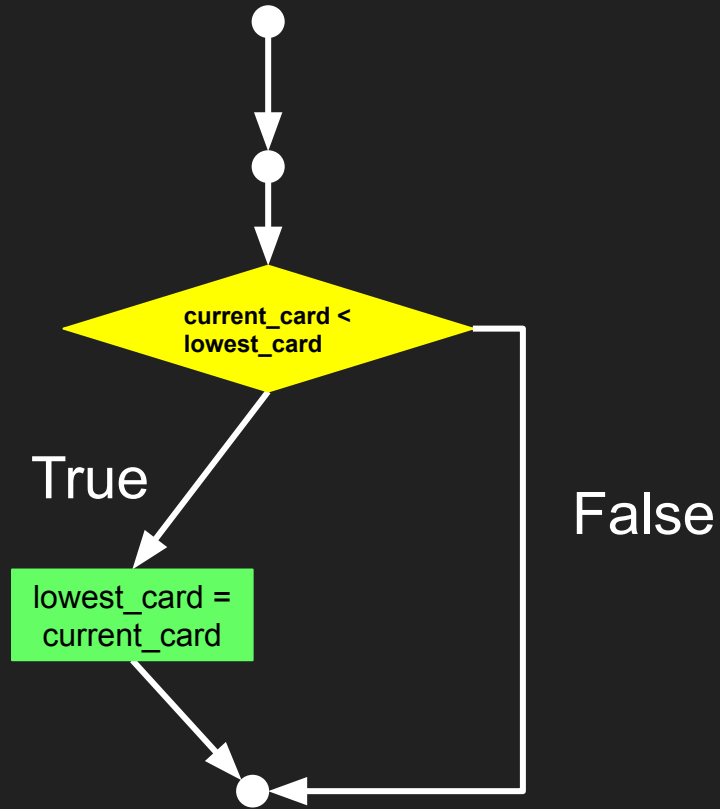
Finding the low card pseudocode:

1 lowest_card = first card in deck

2 Repeatedly until end of deck:

3 if current_card < lowest_card:

4 lowest_card = current_card



Loops

- Used to execute statements in a program repeatedly, an arbitrary number of times
 - Asking the computer: “Run this code, over and over, until a certain condition is False”
- Loops as many times as needed (but make sure it’s finite – the loop must end for the program to continue!)
 - “Infinite loop”: a loop that never ends

Loops

- Used to execute statements in a program repeatedly, an arbitrary number of times

Finding the low card pseudocode:

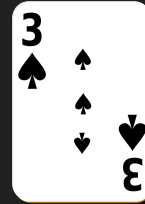
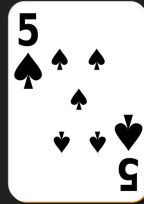
1 `lowest_card = first card in deck`

2 **Repeatedly until end of deck:**

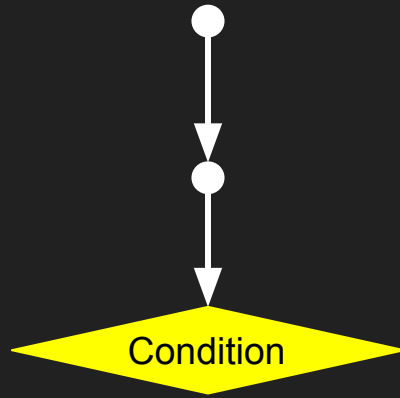
3 **if** `current_card < lowest_card:`

4 `lowest_card = current_card`

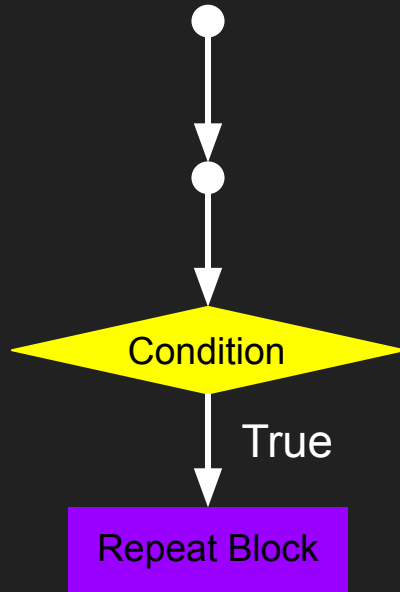
Loop



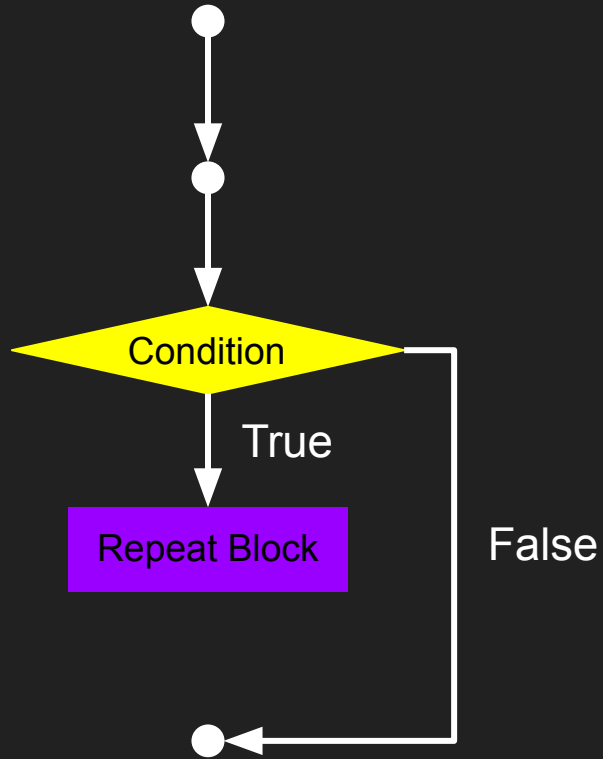
Loops



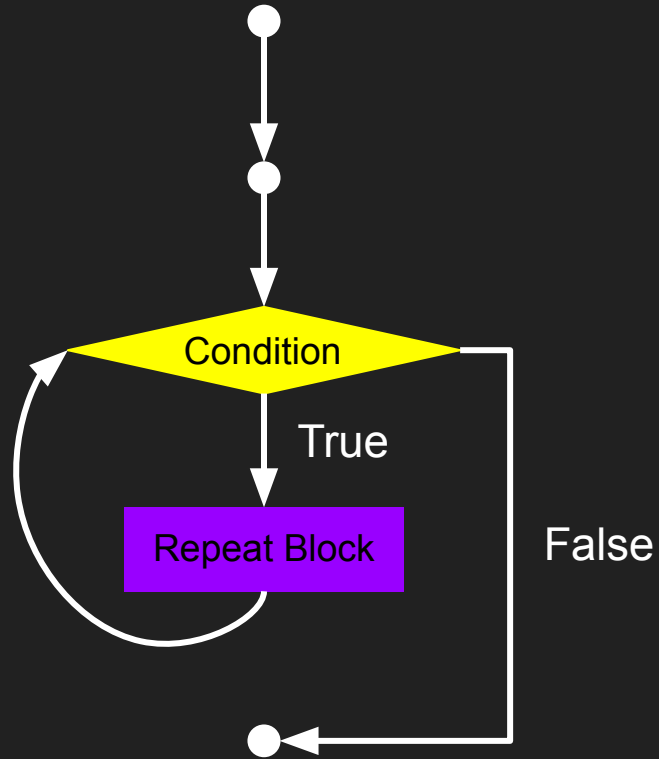
Loops



Loops

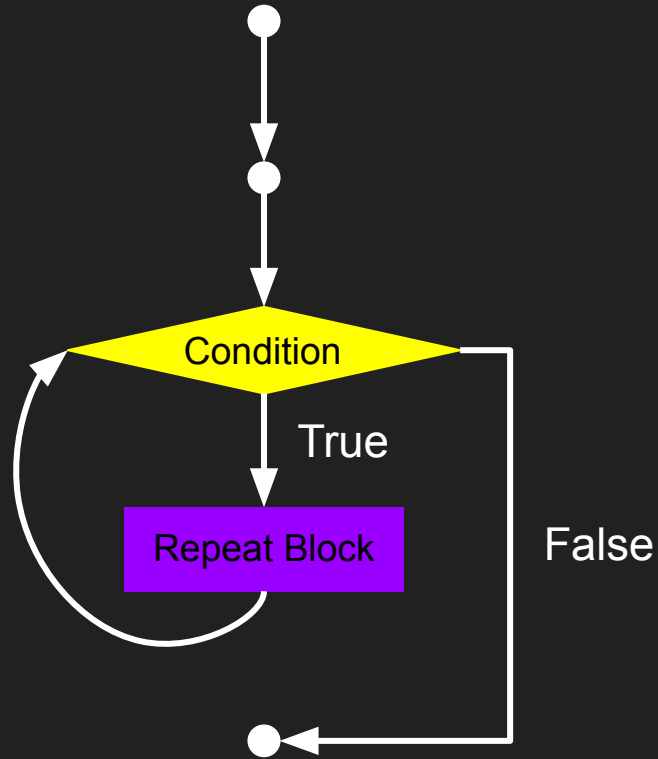


Loops



“While” Loops

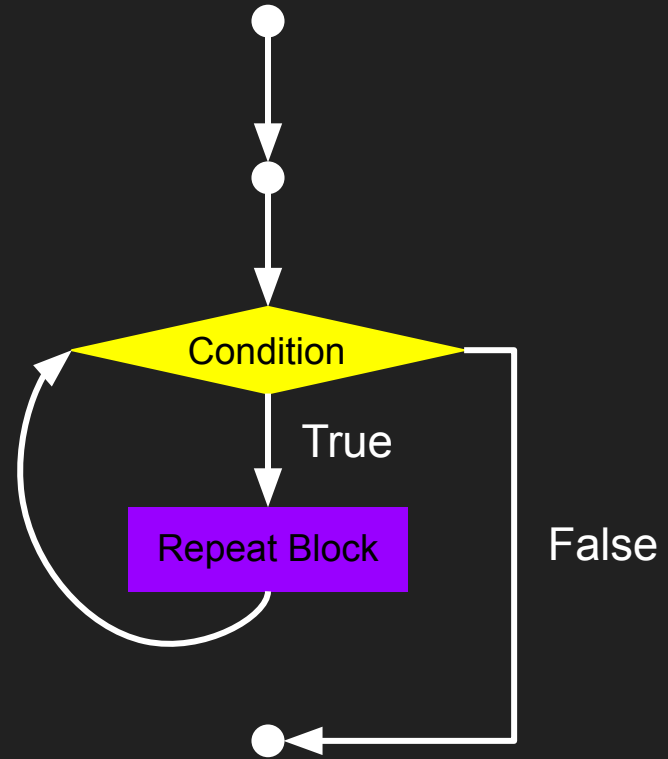
Repeat while
condition is true



Loops

Finding the low card pseudocode:

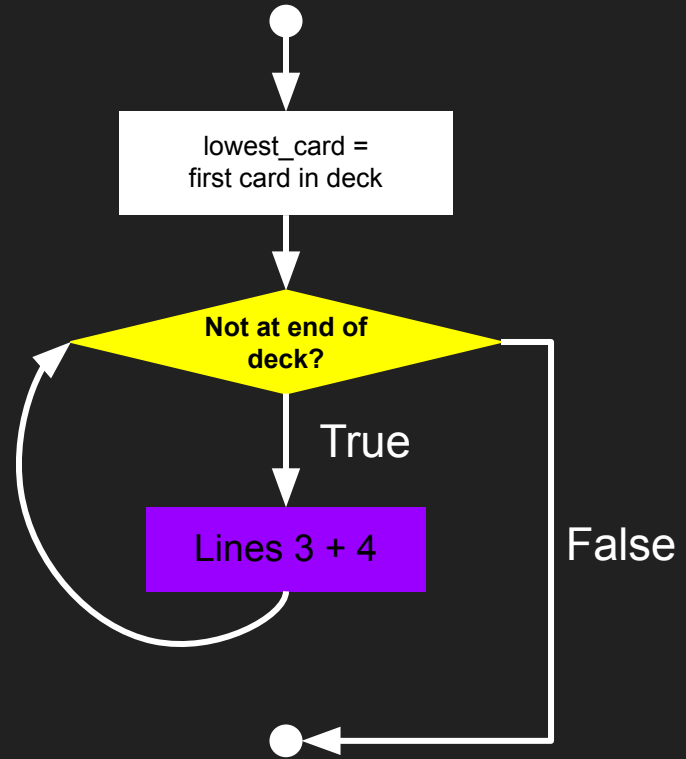
- 1 lowest_card = first card in deck
- 2 Repeatedly until end of deck:
- 3 if current_card < lowest_card:
- 4 lowest_card = current_card



Loops

Finding the low card pseudocode:

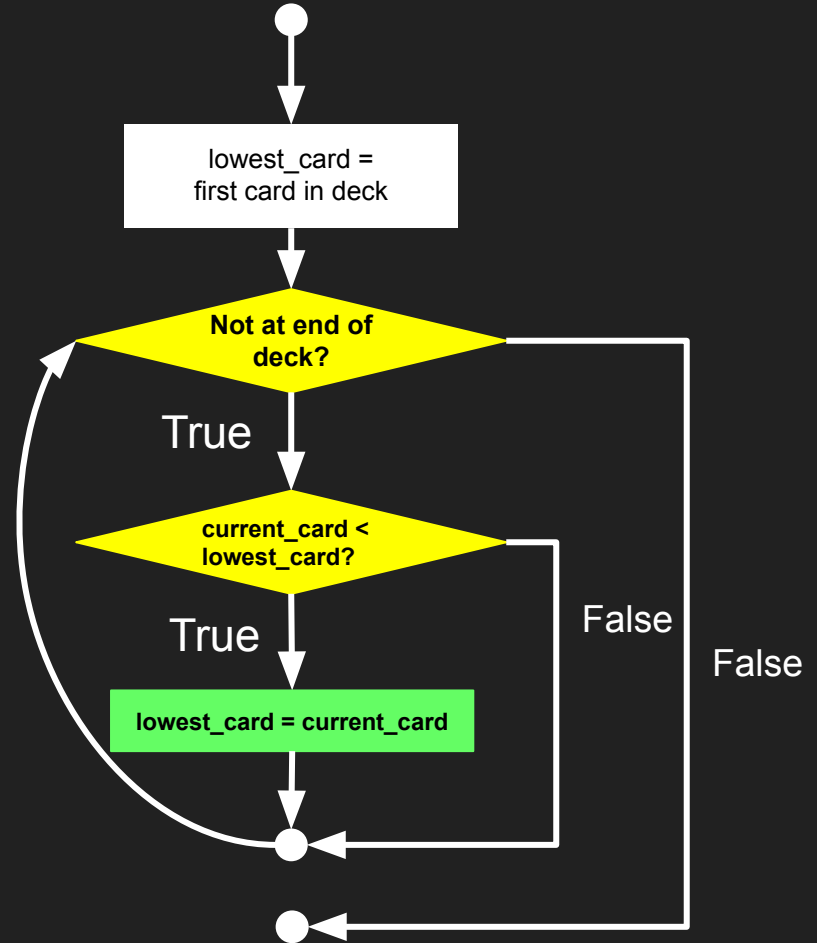
- 1 `lowest_card = first card in deck`
- 2 Repeatedly until end of deck:
- 3 if `current_card < lowest_card`:
- 4 `lowest_card = current_card`



Loops

Finding the low card pseudocode:

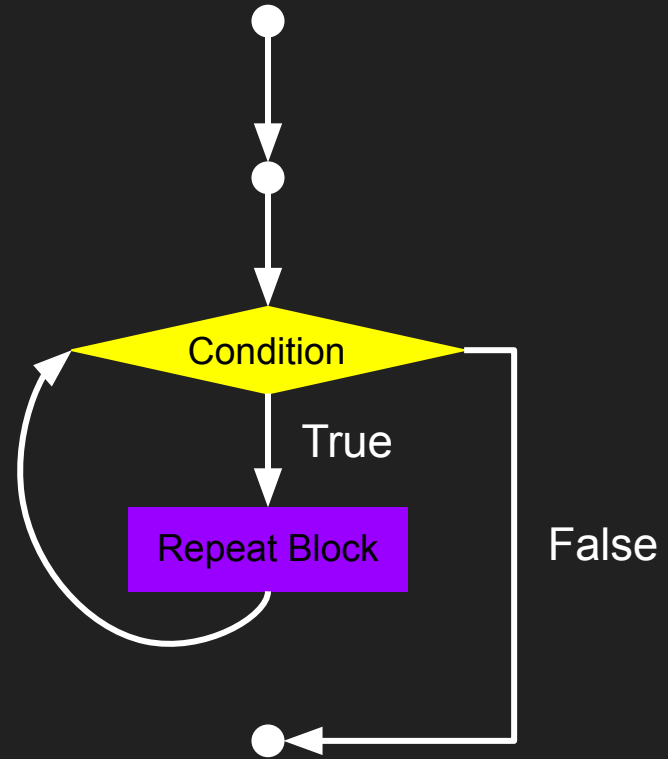
- 1 `lowest_card = first card in deck`
- 2 Repeatedly until end of deck:
- 3 if `current_card < lowest_card`:
- 4 `lowest_card = current_card`



Syntax

while **<condition>**:

<repeat action>



Examples

while **counter** <= 4:

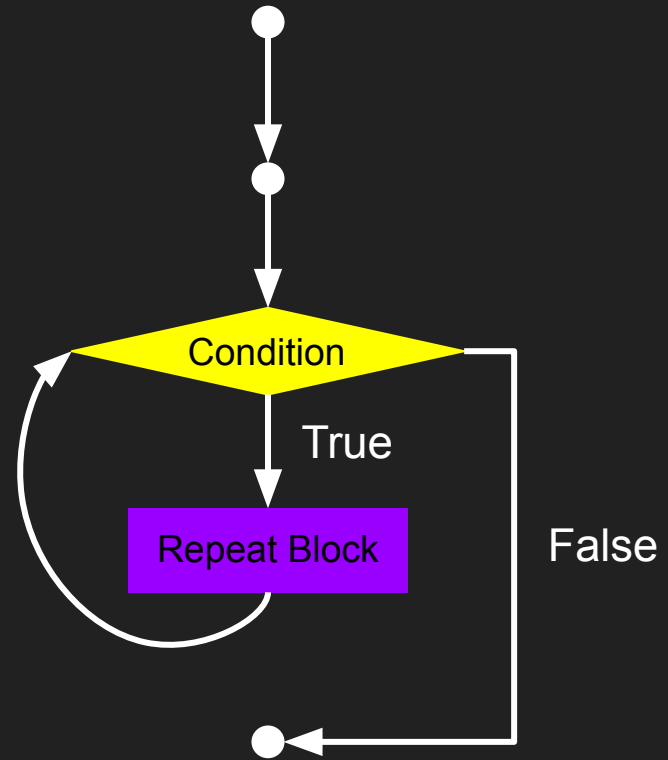
<do something>

while **emotion** == "happy":

<do something>


while **not finished**:

<do something>



Practice Memory Diagram

```
1  def loop(stop: int) -> None:
2      condition: bool = True
3      num_loops: int = 0
4      while condition:
5          print(num_loops)
6          num_loops = num_loops + 1
7          if num_loops >= stop:
8              condition = False
9
10 loop(stop=2)
```

A vertical line is drawn on the left side of the code, spanning from line 2 to line 9. A curved arrow starts at the right side of line 8, loops back to the left side of line 5, and ends with a downward-pointing arrowhead, indicating the loop's continuation.

Practice Memory Diagram

```
1  def characters(msg: str) -> None:
2      index: int = 0
3      while index < len(msg):
4          print(msg[index])
5          index = index + 1
6
7  characters(msg="Howdy")
```

Bonus Lesson: Relative Reassignment Operators

Reassigning a variable relative to its current value: `i = i + 1`

Addition reassignment operator shorthand has the same effect: `i += 1`

Since you will use meaningfully descriptive variable names, this is a big improvement!

`total_dollars = total_dollars + next_donation` vs `total_dollars += next_donation`

```
1 def characters(msg: str) -> None:
2     index: int = 0
3     while index < len(msg):
4         print(msg[index])
5         index = index + 1
6
7 characters(msg="Howdy")
```

```
1 def characters(msg: str) -> None:
2     index: int = 0
3     while index < len(msg):
4         print(msg[index])
5         index += 1
6
7 characters(msg="Howdy")
```

Before	After
<code>i = i + expr</code>	<code>i += expr</code>
<code>i = i - expr</code>	<code>i -= expr</code>
<code>i = i * expr</code>	<code>i *= expr</code>
<code>i = i / expr</code>	<code>i /= expr</code>
<code>i = i % expr</code>	<code>i %= expr</code>
<code>i = i // expr</code>	<code>i //= expr</code>
<code>i = i ** expr</code>	<code>i **= expr</code>