Conditional Execution

Conditional Steps with if

- We can use if statements to make choices based on if some condition is True or False

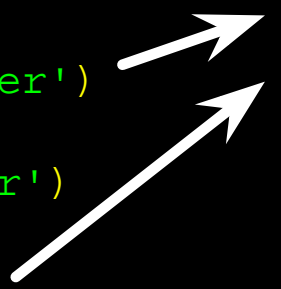
Program:

```
x = 5
if x < 10:
    print('Smaller')
if x > 20:
    print('Bigger')

print('Done')
```

Output:

```
Smaller
Done
```



Comparison Operators

- Boolean expressions ask a question and produce a Yes or No result which we use to control program flow
- Boolean expressions using comparison operators evaluate to True / False or Yes / No
- Comparison operators look at variables but do not change the variables

Python	Meaning
<	Less than
<=	Less than or Equal to
==	Equal to
>=	Greater than or Equal to
>	Greater than
!=	Not equal

Remember: “=” is used for assignment.

Comparison Operators

```
x = 5
if x == 5 :
    print('Equals 5')
if x > 4 :
   print('Greater than 4')
if  x >= 5 :
    print('Greater than or Equals 5')
if x < 6 : print('Less than 6')
if x <= 5 :
    print('Less than or Equals 5')
if x != 6 :
    print('Not equal 6')
```

Equals 5

Greater than 4

Greater than or Equals 5

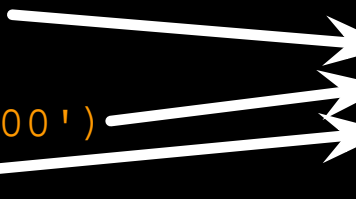
Less than 6

Less than or Equals 5

Not equal 6

Nested Conditionals

```
x = 42
if x > 1 :
    print('More than one')
    if x < 100 :
        print('Less than 100')
print('All done')
```



Output:

More than one
Less than 100
All done

The else statement

- Sometimes we want to do one thing if a logical expression is true and something different if it does not satisfy the comparison

```
x = 2

if x > 2 :
    print('Bigger')
else :
    print('Smaller')

print('All done')
```

Output:

```
Smaller
All done
```

More Conditional Structures…

The else-if statement (elif)

- The elif statement will **only** check the conditional if the previous if or elif statement was **false**

```
x = 0
if x < 2 :
    print('small')
elif x < 10 :
    print('Medium')
else :
    print('LARGE')
print('All done')
```

Output:

small
All done

The else-if statement (elif)

- The elif statement will only check the conditional if the previous if or elif statement was false

```
x = 8
if x < 2 :
    print('small')
elif x < 10 :
    print('Medium')
else :
    print('LARGE')
print('All done')
```

Output:

Medium
All done

What is the output of:

```
month = 9

if month < 10:
    month = month - 3
elif month >= 10:
    month = month - 5

print(month)
```

You can use elif without else

```
# No Else
x = 5
if x < 2 :
    print('Small')
elif x < 10 :
    print('Medium')

print('All done')
```

Output:

```
Small
All done
```

Review (True or False)

1. The `elif` statement can be used multiple times within an `if-elif-else` block.

2. The = operator is used to compare two values for equality, while the == operator is used to assign a value to a variable.

3. `bool("False")` evaluates to False.

Review (True or False)

1. The `elif` statement can be used multiple times within an `if-elif-else` block.
2. The = operator is used to compare two values for equality, while the == operator is used to assign a value to a variable.
3. `bool("False")` evaluates to False.

Loops and Iteration

Loops

Iterating over a set of items…

(Indefinite) Loops

- Sometimes we want to continue doing something (looping) until a certain condition is met
- We might not know how long it takes until the condition is met, so the loop should run forever (until we tell it to stop)
- We can write a loop that runs forever using the Python while construct
- These loops are called “indefinite loops” because they execute an unlimited number of times

A Simple Indefinite Loop

```
counter = 0
while counter < 5:
    print(counter)
    counter = counter + 1

print(Done!')
```

0
1
2
3
4
Done!

A Simple Indefinite Loop

```
counter = 0
while counter < 5:
    print(counter)
    counter = counter + 1

print(Done!')
```

```
0
0
0
0
0
...
0
...
```

(Definite) Loops

- In another case, we have a list of items that has an end, a finite set of things
- We can write a loop to run the loop once for each of the items in a set using the Python for construct
- These loops are called “definite loops” because they execute an exact number of times

A Simple Definite Loop

```
for i in [5, 4, 3, 2, 1] :
    print(i)
print('Blastoff!')
```

```
5
4
3
2
1
Blastoff!
```

A Definite Loop with Strings

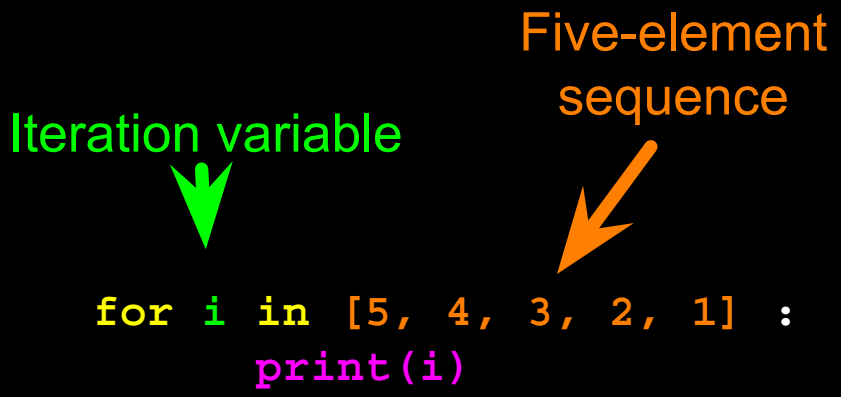
```
friends = ['Joseph', 'Glenn', 'Sally']
for friend in friends :
    print('Happy New Year:', friend)
print('Done!')
```

```
Happy New Year: Joseph
Happy New Year: Glenn
Happy New Year: Sally
```

```
Done!
```

Looking at in...

- The iteration variable "iterates" through the sequence (ordered set)
- The block (body) of code is executed once for each value in the sequence
- The iteration variable moves through all of the values in the sequence



Making “smart” loops

The trick is “knowing” something about the whole loop when you are stuck writing code that only sees one entry at a time

Set some variables to initial values

for thing in data:

Look for something or do something to each entry separately, updating a variable

Look at the variables

Looping Through a Set

```
print('Before')
for thing in [9, 41, 12, 3, 74, 15] :
     print(thing)
print('After')
```

```
$ python basicloop.py
Before
9
41
12
3
74
15
After
```

What is the Largest Number?

What is the Largest Number?

3

What is the Largest Number?

41

What is the Largest Number?

12

What is the Largest Number?

9

What is the Largest Number?

74

What is the Largest Number?

15

What is the Largest Number?

What is the Largest Number?

3 41 12 9 74 15

What is the Largest Number?

largest_so_far -1

What is the Largest Number?

3

largest_so_far 3

What is the Largest Number?

41

largest_so_far 41

What is the Largest Number?

12

largest_so_far 41

What is the Largest Number?

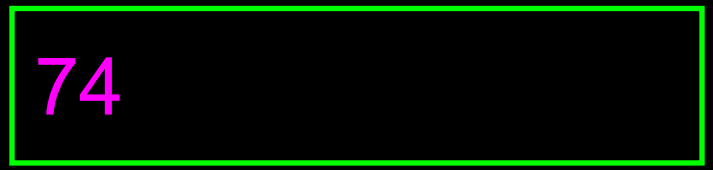
9

largest_so_far 41

What is the Largest Number?

74

largest_so_far



What is the Largest Number?

15

74

What is the Largest Number?

3 41 12 9 74 15

74

Finding the Largest Value

```
largest_so_far = -1
print('Before', largest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num > largest_so_far :
        largest_so_far = the_num
    print(largest_so_far, the_num)

print('After', largest_so_far)
```

```
$ python largest.py
Before -1
9 9
41 41
41 12
41 3
74 74
74 15
After 74
```

We make a variable that contains the largest value we have seen so far. If the current number we are looking at is larger, it is the new largest value we have seen so far.

More Loop Patterns…

Counting in a Loop

```
counter = 0

print('Before', counter)

for thing in [9, 41, 12, 3, 74, 15] :
    counter = counter + 1
    print(counter, thing)

print('After', counter)
```

```
$ python countloop.py
Before 0
1 9
2 41
3 12
4 3
5 74
6 15
After 6
```

To count how many times we execute a loop, we introduce a counter variable that starts at 0 and we add one to it each time through the loop.

What is the output of:

```
sum = 0
for thing in [9, 41, 12, 3, 74, 15] :
    sum = sum + thing
print(sum)
```

Filtering in a Loop

```
print('Before')
for value in [9, 41, 12, 3, 74, 15] :
    PRINT OUT ALL NUMBERS > 20
    ...
print('After')
```

Filtering in a Loop

```
print('Before')
for value in [9, 41, 12, 3, 74, 15] :
    if value > 20:
        print('Large number',value)
print('After')
```

```
$ python search1.py
Before
Large number 41
Large number 74
After
```

We use an if statement in the loop to catch / filter the values we are looking for.

Finding the Smallest Value

```
smallest_so_far = -1
print('Before', smallest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num < smallest_so_far :
        smallest_so_far = the_num
    print(smallest_so_far, the_num)

print('After', smallest_so_far)
```

We switched the variable name to smallest_so_far and switched the > to <

Finding the Smallest Value

```
smallest = None
print('Before')
for value in [9, 41, 12, 3, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print(smallest, value)
print('After', smallest)
```

```
$ python smallest.py
Before
9 9
9 41
9 12
3 3
3 74
3 15
After 3
```

We still have a variable that is the smallest so far. The first time through the loop smallest is None, so we take the first value to be the smallest.

The is and is not Operators

```
smallest = None
print('Before')
for value in [3, 41, 12, 9, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print(smallest, value)

print('After', smallest)
```

- Python has an is operator that can be used in logical expressions
- Implies “is the same as”
- Similar to, but stronger than ==
- is not also is a logical operator

Acknowledgements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

Adapted by Minh Tran, University of Chicago, Department of Computer Science for use in the pre-college course Creating Computer Games for Learning (Summer 2026).