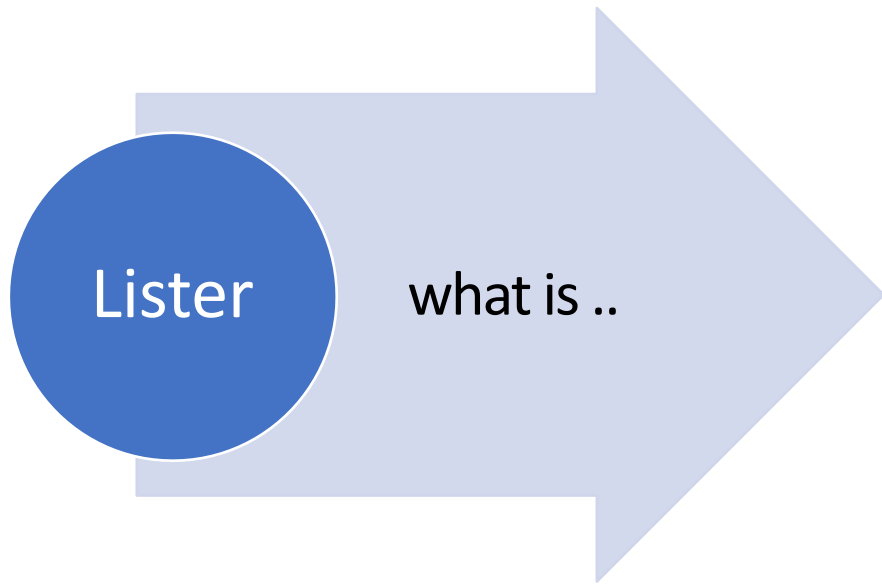


3**INTRODUCING LISTS****33**

What Is a List?	33
Accessing Elements in a List	34
Index Positions Start at 0, Not 1	35
Using Individual Values from a List	35
<i>Exercise 3-1: Names</i>	36
<i>Exercise 3-2: Greetings</i>	36
<i>Exercise 3-3: Your Own List</i>	36
Changing, Adding, and Removing Elements	36
Modifying Elements in a List	36
Adding Elements to a List	37
Removing Elements from a List	38
<i>Exercise 3-4: Guest List</i>	42
<i>Exercise 3-5: Changing Guest List</i>	42
<i>Exercise 3-6: More Guests</i>	42
<i>Exercise 3-7: Shrinking Guest List</i>	43
Organizing a List	43
Sorting a List Permanently with the <code>sort()</code> Method	43
Sorting a List Temporarily with the <code>sorted()</code> Function	44
Printing a List in Reverse Order	45
Finding the Length of a List	45
<i>Exercise 3-8: Seeing the World</i>	46
<i>Exercise 3-9: Dinner Guests</i>	46
<i>Exercise 3-10: Every Function</i>	46



- A list is a **collection** of items in a **particular** order.
- You can put **anything** you want into a list, and the items in your list **don't have to be related** in any particular way

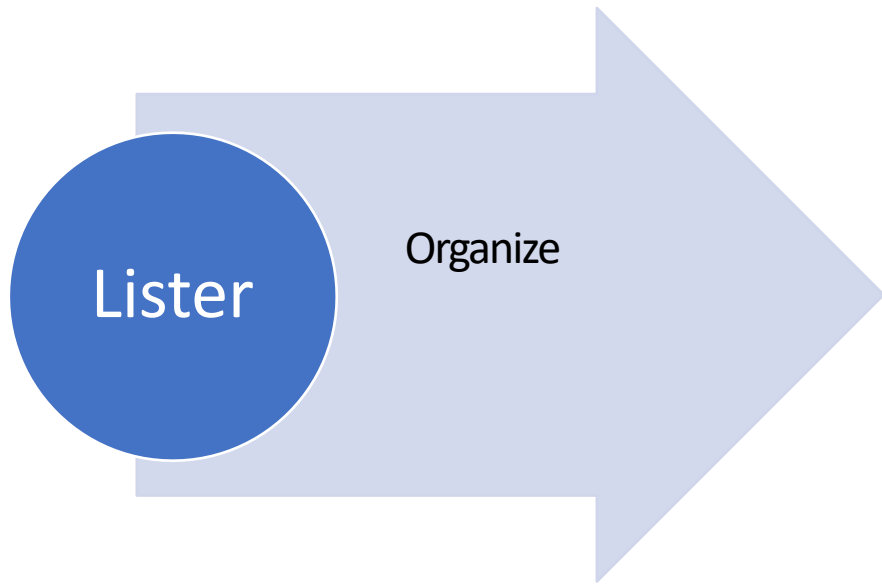


Lister

change
add
remove

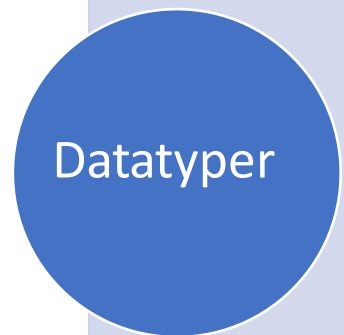
- To change an element, use the **name** of the list followed by the **index** of the element you want to change, and then provide the new **value** you want that item to have
- You can add a new element at any position in your list by using the **insert()** method.
- The **append()** method makes it easy to build lists dynamically.
- You can remove an item from
 - any position in a list using the **del** statement if you know its index
 - The **pop()** method removes the last item in a list
 - If you only know the value of the item you want to remove, you can use the **remove()** method.

```
In [ ]: 1
```



- Python's **sort()** method makes it easy to sort a list.
 - default alphabetically
 - `sort(reverse=True)`
- To maintain the **original order** of a list but present it in a sorted order, you can use the **sorted()** function.
- To reverse the original order of a list, you can use the **reverse()** method
- You can find the length of a list by using the **len()** function.

```
In [ ]: 1
```



Lister –
følger
bogen

4	WORKING WITH LISTS	49
Looping Through an Entire List		49
A Closer Look at Looping		50
Doing More Work Within a for Loop		51
Doing Something After a for Loop		52
Avoiding Indentation Errors		53
Forgetting to Indent		53
Forgetting to Indent Additional Lines		54
Indenting Unnecessarily		55
Indenting Unnecessarily After the Loop		55
Forgetting the Colon		56
<i>Exercise 4-1: Pizzas</i>		56
<i>Exercise 4-2: Animals</i>		56
Making Numerical Lists		57
Using the range() Function		57
Using range() to Make a List of Numbers		58
Simple Statistics with a List of Numbers		59
List Comprehensions		59
<i>Exercise 4-3: Counting to Twenty</i>		60
<i>Exercise 4-4: One Million</i>		60
<i>Exercise 4-5: Summing a Million</i>		60
<i>Exercise 4-6: Odd Numbers</i>		60
<i>Exercise 4-7: Threes</i>		60
<i>Exercise 4-8: Cubes</i>		60
<i>Exercise 4-9: Cube Comprehension</i>		60
Working with Part of a List		61
Slicing a List		61
Looping Through a Slice		62
Copying a List		63
<i>Exercise 4-10: Slices</i>		65
<i>Exercise 4-11: My Pizzas, Your Pizzas</i>		65
<i>Exercise 4-12: More Loops</i>		65



Datatyper

Lister –
Looping

```
magicians = ['alice', 'david', 'carolina']  
for magician in magicians:  
    print(magician)
```

4 WORKING WITH LISTS

49

Looping Through an Entire List	49
A Closer Look at Looping	50
Doing More Work Within a for Loop	51
Doing Something After a for Loop	52

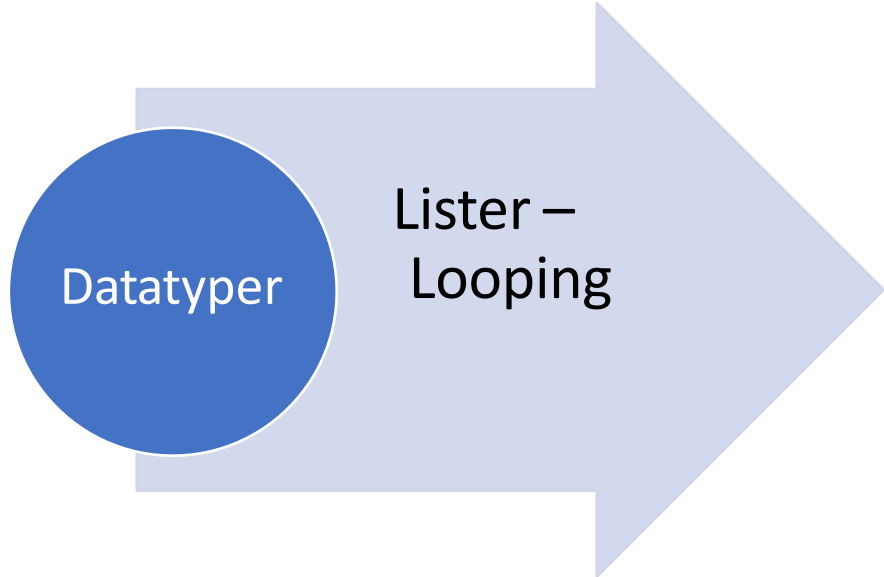
Looping allows you to take the **same action**, or set of actions, with every item in a list. As a result, you'll be able to work efficiently with lists **of any length**, including those with thousands or even millions of items



In []: 1

When you're using loops for the first time, keep in mind that the set of steps is **repeated once for each item** in the list

Python uses **indentation** to determine how a line, or group of lines, is related to the rest of the program



Looping Through an Entire List	49
A Closer Look at Looping	50
Doing More Work Within a for Loop	51
Doing Something After a for Loop	52
Avoiding Indentation Errors	53
Forgetting to Indent	53
Forgetting to Indent Additional Lines	54
Indenting Unnecessarily	55
Indenting Unnecessarily After the Loop	55
Forgetting the Colon	56
Exercise 4-1: Pizzas	56
Exercise 4-2: Animals	56

```
In [ ]: 1
```

NOTE

Sometimes your loop will run without any errors but won't produce the expected result.

```
In [ ]: 1
```

4-2. Animals:
print out the n

```
In [ ]: 1
```

Datatyper

Lister – Looping

```
squares = []  
for value in range(1, 11):  
    square = value ** 2  
    squares.append(square)  
  
print(squares)
```

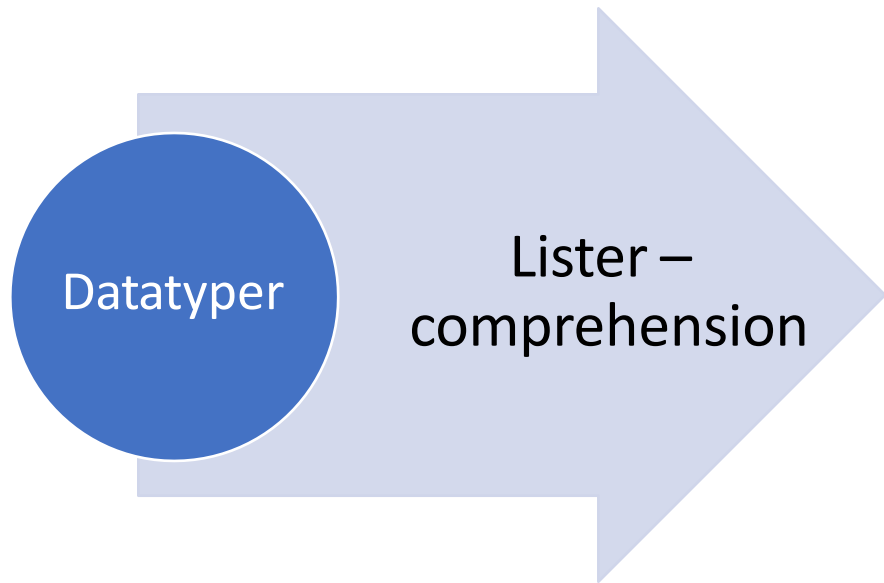
Vigtigt eksempel!

```
In [ ]: 1
```

Making Numerical Lists.	57
Using the range() Function	57
Using range() to Make a List of Numbers	58
Simple Statistics with a List of Numbers	59
List Comprehensions	59
Exercise 4-3: Counting to Twenty	60
Exercise 4-4: One Million.	60

Lists are ideal for storing **sets of numbers**, and Python provides a variety of **tools** to help you work efficiently with lists of numbers.

- the **range()** function to print a series of numbers
- Using range() to Make a **List of Numbers**
- use the range() function to tell Python to **skip** numbers in agiven range



Making Numerical Lists.	57
Using the range() Function	57
Using range() to Make a List of Numbers	58
Simple Statistics with a List of Numbers	59
List Comprehensions	59
Exercise 4-3: Counting to Twenty	60
Exercise 4-4: One Million.	60

A list comprehension **combines** the **for** loop and the creation of new elements into one line, and **automatically** appends each new element

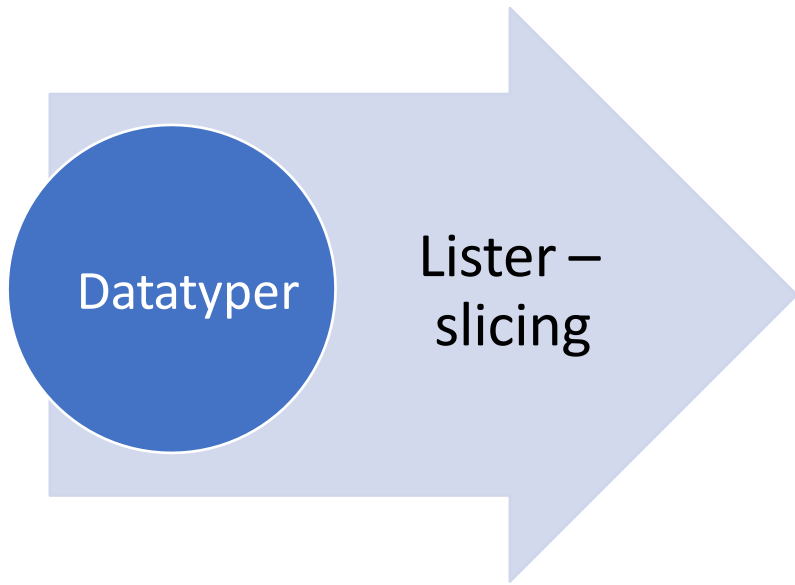
```
1 bikemsg = []
2 for bike in mybikes:
3     msg=f"Her er min {bike}"
4     bikemsg.append(msg)
```



```
1 bikemsg = [f"Her er min {bike}" for bike in mybikes]
```

4-3. Counting to Twenty: Use a for loop

```
1 mynumbers=range(1,21)
```



Working with Part of a List	61
Slicing a List	61
Looping Through a Slice	62
Copying a List	63
Exercise 4-10: Slices	65
Exercise 4-11: My Pizzas, Your Pizzas.	65
Exercise 4-12: More Loops.	65

To make a **slice**, you specify the index of the first and last elements you want to work with. Python stops one item **before** the second index you specify

- `players[0:3]`
- `players[:4]`
- `players[2:]`
- `players[-3:]`

```
In [ ]: 1
```

Datatypes

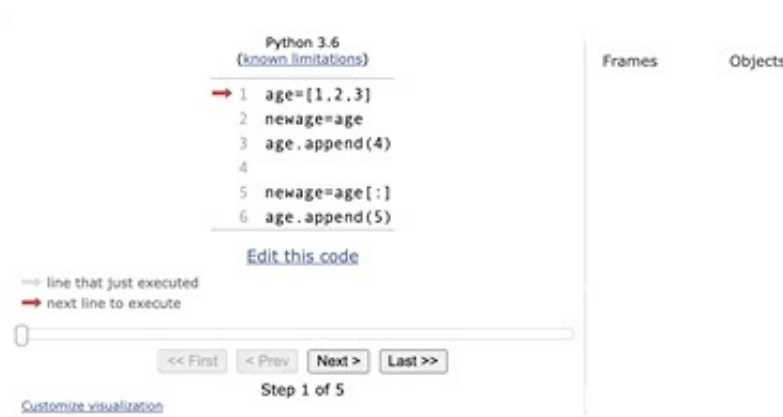
List – slicing

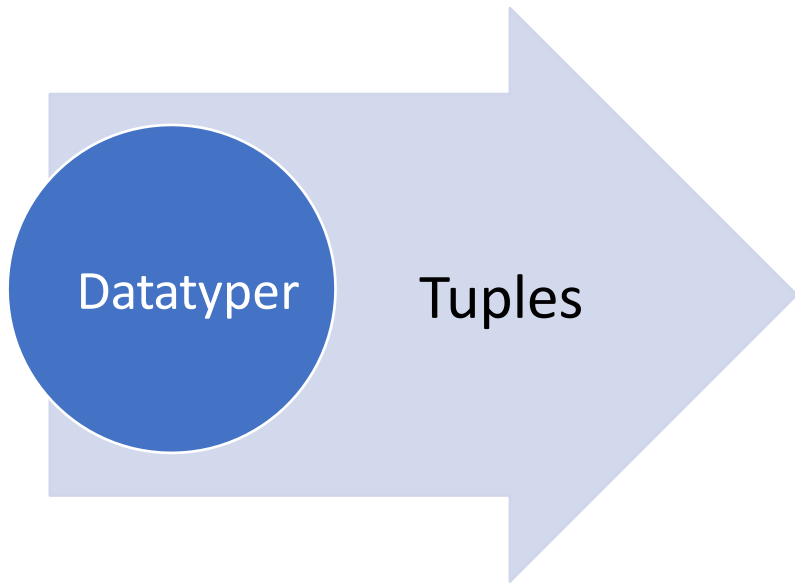
Working with Part of a List	61
Slicing a List	61
Looping Through a Slice	62
Copying a List	63
Exercise 4-10: Slices	65
Exercise 4-11: My Pizzas, Your Pizzas	65
Exercise 4-12: More Loops	65

To copy a list, you can make a slice that includes the entire original list by omitting the first index and the second index (`[:]`)

NOTE

Basically, if you're trying to work with a copy of a list and you see unexpected behavior, make sure you are copying the list using a slice





A tuple looks just like a list except you use **parentheses** instead of square brackets.

Tuples are **immutable**. Use them when you want to store a set of values that should **not** be changed throughout the life of a program

```
1 size=(600,400)
```

```
1 size[0]
```

```
600
```

```
1 size[0]=122
```

```
-----  
TypeError
```

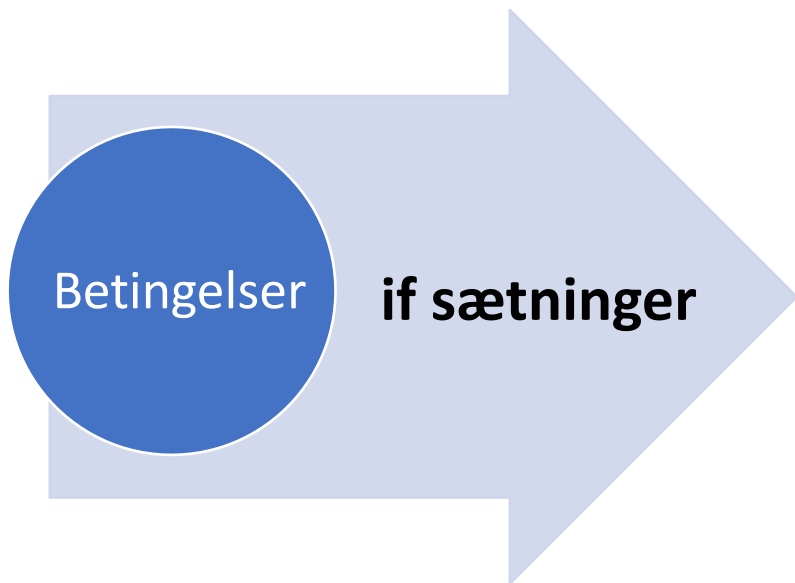


Betingelser



if sætninger

5	71
IF STATEMENTS	
A Simple Example	72
Conditional Tests	72
Checking for Equality	72
Ignoring Case When Checking for Equality	73
Checking for Inequality	74
Numerical Comparisons	74
Checking Multiple Conditions	75
Checking Whether a Value Is in a List	76
Checking Whether a Value Is Not in a List	77
Boolean Expressions	77
<i>Exercise 5-1: Conditional Tests</i>	78
<i>Exercise 5-2: More Conditional Tests</i>	78
if Statements	78
Simple if Statements	78
if-else Statements	79
The if-elif-else Chain	80
Using Multiple elif Blocks	82
Omitting the else Block	82
Testing Multiple Conditions	83
<i>Exercise 5-3: Alien Colors #1</i>	84
<i>Exercise 5-4: Alien Colors #2</i>	84
<i>Exercise 5-5: Alien Colors #3</i>	85
<i>Exercise 5-6: Stages of Life</i>	85
<i>Exercise 5-7: Favorite Fruit</i>	85
Using if Statements with Lists	85
Checking for Special Items	86
Checking That a List Is Not Empty	87
Using Multiple Lists	88
<i>Exercise 5-8: Hello Admin</i>	89
<i>Exercise 5-9: No Users</i>	89
<i>Exercise 5-10: Checking Usernames</i>	89
<i>Exercise 5-11: Ordinal Numbers</i>	89
Styling Your if Statements	90
<i>Exercise 5-12: Styling if statements</i>	90
<i>Exercise 5-13: Your Ideas</i>	90



Boolean values - True, False - provide an efficient way to track the **state** of a program or a particular condition that is important in your program.

5

IF STATEMENTS

71

A Simple Example	72
Conditional Tests	72
Checking for Equality	72
Ignoring Case When Checking for Equality	73
Checking for Inequality	74
Numerical Comparisons	74
Checking Multiple Conditions	75
Checking Whether a Value Is in a List	76
Checking Whether a Value Is Not in a List	77
Boolean Expressions	77
Exercise 5-1: Conditional Tests	78
Exercise 5-2: More Conditional Tests	78

At the heart of **every if statement** is an **expression** that can be **evaluated** as **True or False** and is called a conditional test.

- Single condition:
 - if age == 18:
 - if age > 18:
 - if age < 18:
 - if age != 18:
- Multiple conditions
 - if (age > 67) **or** (age < 18):
 - if (age > 67) **and** (age < 18):
- Element in list
 - if "Trek" **in** mybikes:
 - if "Trek" **not in** mybikes:

Betingelser

if sætninger

5

IF STATEMENTS

71

A Simple Example	72
Conditional Tests	72
Checking for Equality	72
Ignoring Case When Checking for Equality	73
Checking for Inequality	74
Numerical Comparisons	74
Checking Multiple Conditions	75
Checking Whether a Value Is in a List	76
Checking Whether a Value Is Not in a List	77
Boolean Expressions	77
Exercise 5-1: Conditional Tests	78
Exercise 5-2: More Conditional Tests	78

In [22]:

```
1 car="Audi"
2 print(car=="Subaru")
3 print(car.lower()=='audi')
```

False

True

In [34]:

```
1 ages=list(range(0,100,10))
2 ages.append(32)
3 age_lone=21
4 age_birgit=42
5 age_kurt=21
6 print(age_lone > age_birgit)
7 print(age_lone > age_kurt)
8 print(age_lone >= age_kurt)
9 print(f"is 40 in ages? {40 in ages}")
```

False

False

True

is 40 in ages? True



Betingelser

if sætninger

if Statements	78
Simple if Statements	78
if-else Statements	79
The if-elif-else Chain	80
Using Multiple elif Blocks	82
Omitting the else Block	82
Testing Multiple Conditions	83
Exercise 5-3: Alien Colors #1	84
Exercise 5-4: Alien Colors #2	84
Exercise 5-5: Alien Colors #3	85
Exercise 5-6: Stages of Life	85
Exercise 5-7: Favorite Fruit	85

Simple if Statements

The simplest kind of if statement has one test and one action:

```
if conditional_test:
    do something
```



In []: 1

NOTE

Indentation plays the same role in if statements as it did in for loops. **All indented lines** after an if statement **will be executed if the test passes**, and the entire block of indented lines will be ignored if the test does not pass.



Betingelser

if sætninger

if Statements	78
Simple if Statements	78
if-else Statements	79
The if-elif-else Chain	80
Using Multiple elif Blocks	82
Omitting the else Block	82
Testing Multiple Conditions	83
Exercise 5-3: Alien Colors #1	84
Exercise 5-4: Alien Colors #2	84
Exercise 5-5: Alien Colors #3	85
Exercise 5-6: Stages of Life	85
Exercise 5-7: Favorite Fruit	85

Often, you'll want to take one action when a conditional test passes and a different action in all other cases. Python's if-else syntax makes this possible.

```
if condition:
    action
else:
    other action
```

```
if condition:
    action
elif:
    action2
elif:
    action2
else:
    final-action
```

NOTE

Python executes only one block in an if-elif-else chain. It runs each conditional test in order until one passes. When a test passes, the code following that test is executed and Python skips the rest of the tests.

The else block is a catchall statement – so you may want to skip it!



Betingelser



if sætninger

if Statements	78
Simple if Statements	78
if-else Statements	79
The if-elif-else Chain	80
Using Multiple elif Blocks	82
Omitting the else Block	82
Testing Multiple Conditions	83
Exercise 5-3: Alien Colors #1	84
Exercise 5-4: Alien Colors #2	84
Exercise 5-5: Alien Colors #3	85
Exercise 5-6: Stages of Life	85
Exercise 5-7: Favorite Fruit	85

5-3. Alien Colors #1: Imagine an alien was just shot down.

In [41]:

```
1 alien_color='green'
2 if alien_color == 'green':
3     print("you got 5")
```

you got 5



Betingelser



if & lister

Using if Statements with Lists	85
Checking for Special Items	86
Checking That a List Is Not Empty	87
Using Multiple Lists	88
<i>Exercise 5-8: Hello Admin</i>	89
<i>Exercise 5-9: No Users</i>	89
<i>Exercise 5-10: Checking Usernames</i>	89
<i>Exercise 5-11: Ordinal Numbers</i>	89

You can do some interesting work when you combine lists and if statements. You can **watch for special values** that need to be treated differently than other values in the list. You can **manage changing conditions** efficiently, such as the availability of certain items in a restaurant throughout a shift. You can also begin to **prove that your code works** as you expect it to in all possible situations



Betingelser



if & lister

Using if Statements with Lists	85
Checking for Special Items	86
Checking That a List Is Not Empty	87
Using Multiple Lists	88
<i>Exercise 5-8: Hello Admin</i>	89
<i>Exercise 5-9: No Users</i>	89
<i>Exercise 5-10: Checking Usernames</i>	89
<i>Exercise 5-11: Ordinal Numbers</i>	89

5-10. Checking Usernames: Do the following to create a program that simulates how websites ensure that everyone has a unique username.

- Make a list of five or more usernames called `current_users`.
- Make another list of five usernames called `new_users`. Make sure one or two of the new usernames are also in the `current_users` list.
- Loop through the `new_users` list to see if each new username has already been used. If it has, print a message that the person will need to enter a new username. If a username has not been used, print a message saying that the username is available.
- Make sure your comparison is case insensitive. If 'John' has been used, 'JOHN' should not be accepted. (To do this, you'll need to make a copy of `current_users` containing the lowercase versions of all existing users.)