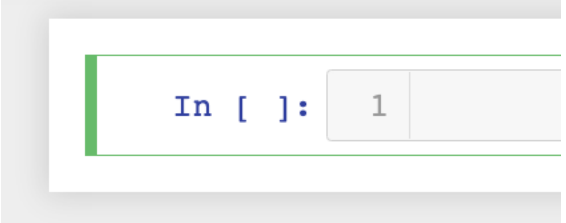
Datatyper

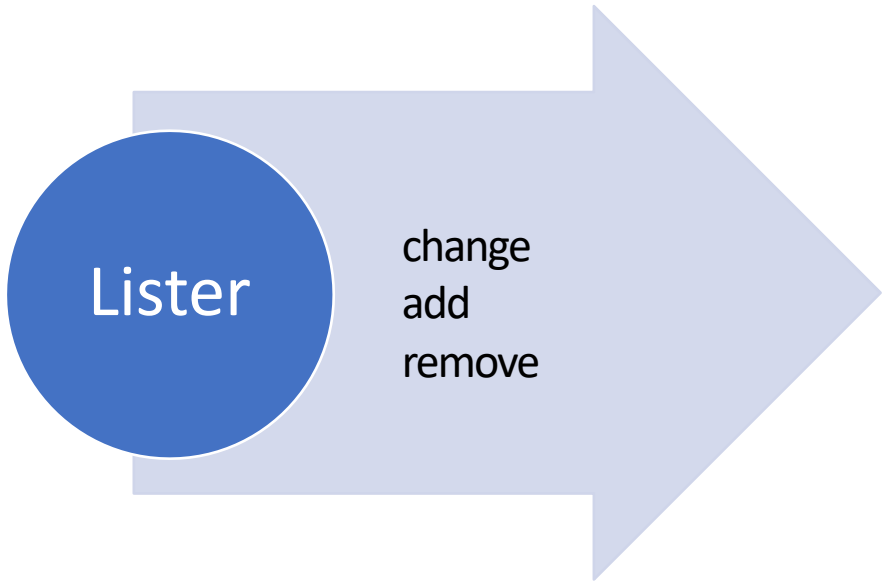
Lister – følger bogen

Lister

what is ..

- 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



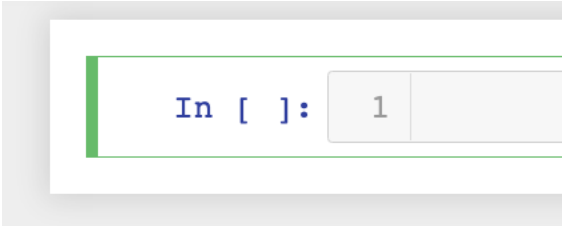


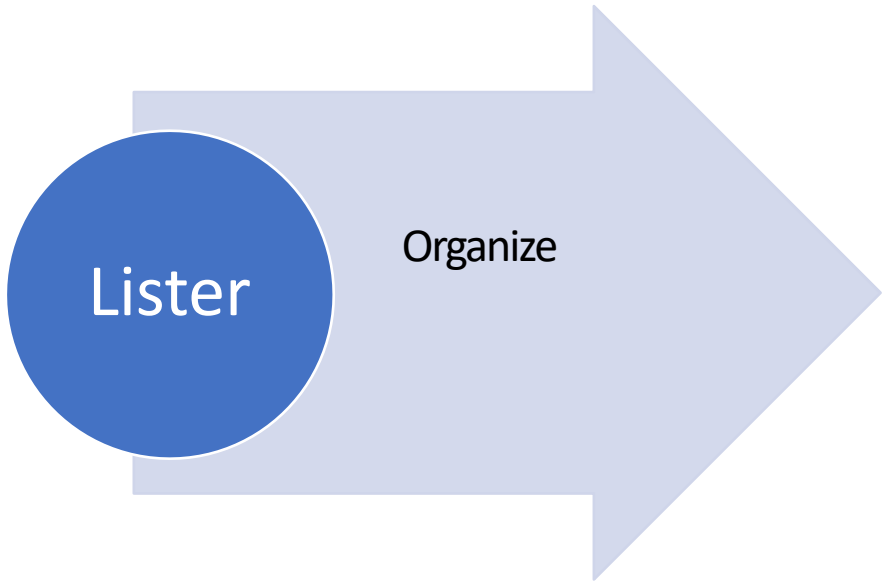
- To change an element, use the **name** of the list followed by the **index** of the element you want to change, and then provide the new **value** you want that item to have

- 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.

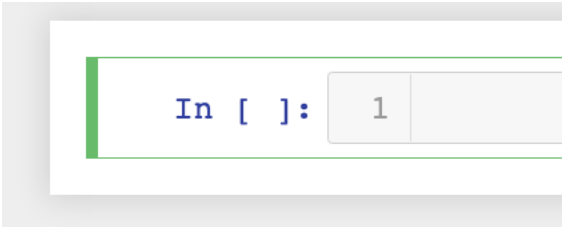
- You can remove an item from
  - any position in a list using the **del** statement if you know its index
  - The **pop**() method removes the last item in a list
  - If you only know the value of the item you want to remove, you can use the **remove**() method.

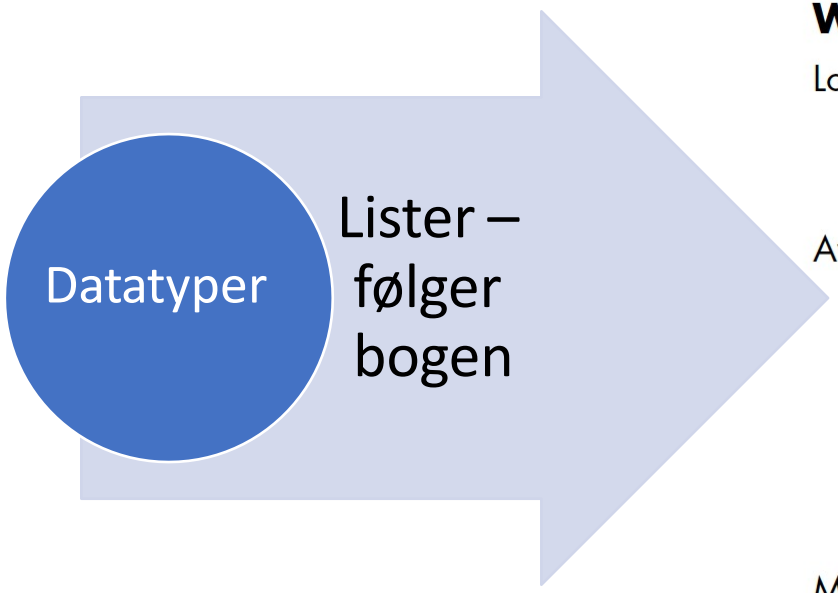




- Python’s **sort**() method makes it easy to sort a list.
  - default alphabtically
  - sort(**reverse**=True)

- 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.





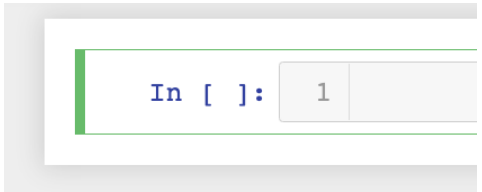
Datatyper

Lister – Looping

4
WORKING WITH LISTS **49**

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

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

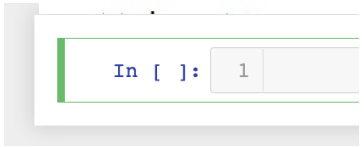


When you're using loops for the first time, keep in mind that the set of steps is **repeated once for each item** in the list

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

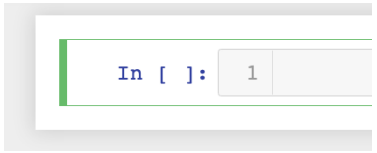
Datatyper

Lister – Looping



NOTE

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



4-2. Animals:
print out the n

Datatyper

Lister – Looping

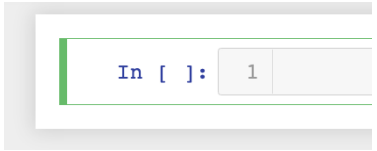
Lists are ideal for storing **sets of numbers**, and Python provides a variety of **tools** to help you work efficiently with lists of numbers.

- 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

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

print(squares)
```

Vigtigt eksempel!



Datatyper

Lister – comprehension

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)
```

Datatyper

Lister – slicing

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:]

Datatyper

Lister – slicing

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

Datatyper

Tuples

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
IF STATEMENTS **71**

Betingelser

if sætninger

5
IF STATEMENTS **71**

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.

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**

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
```

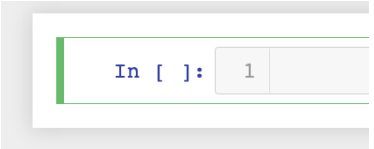
Betingelser

if sætninger

Simple if Statements

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

```
if conditional_test:
    do something
```



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

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

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

In [41]:

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

```
you got 5
```

Betingelser

if & lister

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

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.)