Lecture 1 Notes

4th of October

Lecture 1: Basic Syntax

In this part we will look at the basic syntax of the language, the basics that you will use throughout the whole course.

Printing on the screen

```
print("Hello World")
```

This code will print show on the screen the message : "Hello World". `print` is an example of a function, we will be covering what functions are later on, but there are some functions

Variables and types

Variables are used to store information to be referenced and manipulated in a computer program. They also provide a way of labeling data with a descriptive name, so our programs can be understood more clearly by the reader and ourselves. It is helpful to think of variables as containers that hold information. Their sole purpose is to label and store data in memory. This data can then be used throughout your program.

```
a = 1
b = "Hello"
c = 1.2
d = 'c'
e = False
```

This is how you **define** variables in Python. Python is a **dynamically-typed** language, which means that the **types** of variables don't have to be specified when defining them, and that a variable can change its type and value after being defined. So what are types? Taking the example above, variable `a` is set

to `1`, the type of `1` is and **integer**. There are many different types in computer science but for now we will cover:

- Integers, `a`
- Strings, `b`
- Floats, `c`
- Char, `d`
- Boolean, `e`

```
a = "Hello World"
print(a)
```

In this example we have defined a string variable `a` and set its value to `"Hello World"` and then **parsed** that variable into the `print` function, this will print the same thing as our previous example.

```
a = "Hello World"
print(a)
a = 1
print(a)
```

This is an example showing that Python is dynamically-typed language, as you can see we are **re-defining** variable `a` from a `string` to an `integer`. This will print :

```
Hello World
1
```

We can also convert variables to different types (not always), for example:

```
a = 1
a = str(a)
a = int(a)
```

In this example we use the function `str` to convert the variable a from an integer to a string, then we use the function `int` to convert it back to an integer.

Working with integers and floats

Lets start with the most basic operations:

```
a = 1
b = 5
c = 2
result = (a+b)/c
print(result)
```

As you can see in the example we have defined three integer variables `a`, `b` and `c`, then we have defined a `result` variable which contains the sum of `a` and `b`, divided by c. The `result` variable is of type `float`, this happens because of the `/` (division) operator which in Python returns a float number. This means that the value of `result` is `2.0`. If you wanted to get an integer simply use `//` instead of `/`, fortunately in Python integers and floats are treated similarly, which means that we can add a float and an integer without having to convert any of them.

Operator	Action
+	Addition
-	Substraction
	Multiplication
/	Division
//	Floor Division
	Exponent
%	Modulus (remainder)

These are the basic operators, with the introduction of **libraries** which we will explain further on the arithmetic you can do will expand.

These operators can also be used when we assign a value to a variable. For example:

```
x = 2
x = x*2
```

This declares a variable `x` setting its value to `2`, then updates the value to its current value * 2. Which can also be written as:

```
x = 2
x *= 2
```

The operator `*=` is a combination of `*` and the equal sign we use to assign a value to a variable. The operator, in this case `*`, can be replaced by any of the operators showed before E.g. : `//=`, `+=`, `-=`...

Input and output

If we want to make our program interactive we need to have some sort of input from the user. In this section we will cover the most basic way of getting an

input, through the command line

```
a = input("Hello user! Write whatever you like : ")
print("Here's what you wrote : " + a)
```

The `input` function prints on the screen the argument parsed (we will cover arguments in future lessons) and will be expecting an input from the user, the input will be submitted when the enter key is pressed. The input of the user will be stored into the variable `a` as a string. We will then use the `print` function to show the user whan the input was. The `+` sign is also used for strings, this is called **string-concatenation**, which is when we simply add two strings together into one string.

Challenge

With the knowledge you got from this lecture your task is to create a simple program which asks the user to input **two integers** and then it takes those two numbers and executes at least **3 different operations** on them and outputs the result to the user.V

Hint : keep in mind the different types! What type is an input stored as?

Lecture 2 Notes

March 6, 2020

Lecture 2: Control Flow

In this part of the course we will be looking and boolean login and control flow.

Boolean Variable

The Boolean data type is a data type that has one of two possible values, intended to represent the two truth values of logic and Boolean algebra. The two possible values of a boolean variable are `True` or `False`.

If Statement

An if statement is an operation which checks the condition between variables and executes code if that contition is met. Here is an example:

```
x = True
if x:
    print("Variable x is True!")
else:
    print("Variable x is False!")
```

In this example the variable `x` is checked if it has a value of `True`, if it does the first block of code will be executed. Otherwise the second block of code under `else` will be the one to be executed.

This example simply checks if the variable `x` is `True` or `False`, now we will introduce to more advanced operations

Comparison Operators

Instead of simply checking one boolean variable, using comparison operators it is possible to compare multiple variables and returning a boolean result. Here are the operators used in Python:

```
x == y #x is equal to y
x > y #x is greater than y
```

```
x < y #x is smaller than y
x >= y #x is greater or equal than y
x <= y #x is smaller or euqal than y
```

These are the most important operators, remember than each and everyone of these operations returns a boolean value. For example let's take `x > y`, this operation checsk if the variable `x` is greater than `y`, if `x` is greater than `y` then the operation result will be `True`, otherwise it will be `False`. Here is an example:

```
x = 10
y = 20
if x >= y:
    print("Variable x is greater or equal to y")
else:
    print("Variable x is lower than y")
```

In this case the result of the operation `x >= y` will result in `False` because `x = 10` and `y = 20`, so the block of code which will be run is the one under `else`. The output of this program will be: `Variable x is lower than y`

Elif Statement

It is possible to make multiple separate comparisons, not only `if` and `else`. This is possible using the `elif` statement. Let's look at an example using strings instead of integers:

```
name = "John"
if name == "Bob":
    print("Hi! My name is Bob")
elif name == "John":
    print("Hi! My name is John")
else:
    print("Hi! My name is neither John or Bob")
```

This block of code first checks if the variable `name` is equal to the string `"Bob"`, then it checks if it is equal to the string `"John"`, and in the end if none of prievious were matched, the code under `else` will be executed. The output of this program is `Hi! My name is John`, because the operation `name == "John"` returns `True`.

Not True

If we want to check if something is not true, it is very simple. Here it is:

```
school = "IHGR"
if school != "IHGR":
    print("The school is not IHGR!")
else:
    print("The school is IHGR!")
```

The operator `!=` returns `True` when the two variables are different from each other. In this example the operation: `school != "IHGR"` will return `False` because the variable `school` is equal to the string `"IHGR"`, which means that the output of this program is gonna be `The school is IHGR!`. In Python we can also use the statement `not` to get the opposite value of a boolean result/operation.

```
x = not True
print(x)
```

This program will print `False`.

Or And

Imagine if before execturing a certain block of code you would want to check two different variables, you would have to do this by :

```
name = "Bob"
age = 19
if name == "Bob":
    if age >= 18:
        print("Barman: Here Bob, have a beer")
else:
    print("Barman: Sorry we only serve people named Bob here that are 18+")
```

In this example we are first checking if the variable `name` is equal to the string `"Bob"`, then if it is we are checking if the variable `age` is greater or equal than `18`. We can put this together by using the operator `and`.

```
name = "Bob"
age = 19
if name == "Bob" and age >= 18:
    print("Barman: Here Bob, have a beer")
else:
    print("Barman: Sorry we only serve people named Bob here that are 18+")
```

Now let's say that the "barman" decides to add a new name to the list of people he serves, this is where the operator `or` will help us.

```
name = "John"
age = 19
if (name == "Bob" or name == "John") and age >= 18:
    print("Barman: Here, have a beer")
else:
    print("Barman: Sorry we only serve people named Bob or John here that are 18+")
```

Note that the `or` operation is wrapped into brackets, this because we want to first check if the `name` variable is `"Bob"` or `"John"`, and if one of them is true then we are gonna check the age. The operation inside the brakets returns a boolean result which is then used into the next operation, in this case `and`.

Very Important! :

When using `and`, the operations before and after both have to be `True`.

When using `or`, only one needs to be `True` (both can be, it doesn't matter).

Exercise

Can you tell me what the output of this program is going to be?

```
member = True
age = 41
name = "Lucas"
sport = "Boxing"
if member and age >= 20:
    if (name == "Frank" or name == "Kevin") and sport != "Swimming":
        print(1)
    elif name == "John" or sport == "Boxing":
        print(2)
    else:
        print(3)
elif not member and age > 6 and age < 20:
    if name != "Bob":
        print(4)
    else:
        print(5)
else:
    print(6)
```

And what happens if we change the variables to :

```
member = False
age = 10
name = "Bob"
sport = "Swimming"
```

Challenge

1) Make any program that comes to your mind using as many operators as possible. Ask the user for an input, have some logic to check the input and then process that input however you want and give the user the result.Remember the course material from the previous lesson!
2) Create a profile maker, prompt the user for his name/age/profession etc. Make sure you check each type of input you are given, e.g. make sure the age is always an `int`. Then once you have the user data, save everything to a file. If you forgot how to work with files in python here is something to help you:

```
f = open("myfile.txt", "w")
f.write("Hello World!")
f.close()
```

1) You are given 2 variables: is_bob_home and is_it_night (these variables are either True or False). Write an if-else expression that accounts for all of these variables and follows the following logic:
    1) If bob is home and it is night, print “bob should go to sleep”
    2) If it is not home print “bob should go home” and if it is night print “and then sleep”
    3) if bob is home and it is not night print “bob has nothing to do”

Lecture 3 Notes

November 8, 2019

Lecture 3: Loops and Arrays, Loops and Arrays, Loops. . .

Introduction to Arrays

Arrays are another way to store data. In a nutshell, an array contains a series of variables that are ordered with numbers from 0 to some number (integer) n. Note that if you want to get the 1st element of an array, you must access the 0th element. This is called **0-indexing**. A list can contain multiple different types of variables in Python but this is different in other programming languages.

Syntax of an Array

Some examples of arrays:

```
some_numbers = [1, 2, 3, 4, 5]
some_array = [True, 23.21, "HI!"]
alphabet = ['a', 'b', 'c', 'd',]
```

These arrays can be accessed with '[]', like this:

```
some_numbers[3] # "4th" element in the array => 4
some_array[2]   # => "HI!"
alphabet[0]     # => 'a'
```

So how are arrays useful?

Lets say that we have a program that calculates the average of any number of numbers you give it. So if we give it the numbers 1, 2, and 3, it would give the average of 2. Without arrays, making this would be doable but quite annoying. With arrays, we can just add the number that the user inputs to the end of the array and after the user is done inputting numbers, we take the average.

e.g

In other words: Instead of having

```
user_input_1 = raw_input()
user_input_2 = raw_input()
user_input_3 = raw_input()
user_input_4 = raw_input()
user_input_5 = raw_input()
...
```

We can neatly organise it in an array

```
user_inputs = []
```

Using an array makes the code much more clear and easier to modify.

Advanced Arrays

As already said, arrays are basically a collection of variables in a variable. So what about arrays inside arrays? Array-ception.

Of course, this can be done in Python! These are called **nested arrays**. They can be very useful in many computing problems.

An example of a 2-D array would be:

```
two_dee = [["this", "is a", "nested array"], [1,2,3,4],
    [True, True, False]]
```

A very important thing to realise here is that what a '2-D' list *really* is just a list of lists

Modifying Arrays

In order to add elements to an array you can do

```
myArray = []
myArray.append(1)
myArray.append(4)
myArray.append(41)
# myArray = [1, 4, 41]
myArray.remove(4)
# myArray = [1, 41]
```

'.append(a)' adds a value of 'a' into the array at the end.

'.remove(a)' removes a value 'a' from the array, if it is not found it returns an error.

There is also a function called '.insert(i, elem)' which takes an index of 'i' and puts 'elem' into that location.

Joining Lists

Joining two lists is easy in Python! Here is an example:

```
array1 = ["Hello ", "World! "]
array2 = ["Bye ", "Space!"]
print(array1 + array2)
```

Guess what this program will print.

Very useful interactions with lists

```
arr = [1,3,2,4,5]
len(arr)          # => 5
sum(arr)          # => 15
max(arr)          # => 5
min(arr)          # => 1
arr.sort()        # => [1,2,3,4,5]
arr.pop()         # => [1,2,3,4]
arr.clear()       # => []
```

Slashing and Slicing Lists

Lets say that we have python list like below

```
our_array = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
```

Of course, to get *one* element from the list, e.g 5, we do 'our_array[4]'. But lets say that we want a *range* of values. For example, we could want a *list* of numbers from 2 to 10 from our_array variable. In order to do this we use slicing:

```
our_array[1:10]     # => [2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Similar to the good old array[a] which returns the value at index a but array[a:b] returns the list of values in between the indices of a and b (not including b).

Introduction to Loops

Loops are a way of repeating certain code snippets.

For example the following code snippet would print the numbers from 1 to 99 on the screen.

```
for i in range(0,100):
    print(i)
```

This specific example is a 'for' loop. It takes a variable, in this case i, that changes every time the loop goes on to another iteration.

Note: range is a 'function' that produces a list of number that fulfil $0 \leq x < 100$

While Loops

Another kind of loop would be a 'while' loop. Instead of changing a variable, it takes a statement like in an 'if' statement and loops as long as this statement is true. For example this loop would print "hello" infinitely as the statement is always 'True':

```
while True:
    print("hello")
```

This is usually bad practice in programming as there is no real way to quit the program.

However you can call a keyword called 'break' which essentially breaks the loop and continues with the code below.

```
while True:
    if input() == "":
        break
```

this snippet accepts input but only 'exits' when nothing is given

Loops with arrays

Lets say that you want to take a sum of some array filled with numbers. For this you can use the 'in' keyword.

```
num_array = [...]
total = 0
for num in num_array:
    total += num
print(total)
```

This snippet will take some array called 'num_array' and sum all of it and then print the sum to the user.

The 'in' keyword basically goes through all the element in an array and is very useful compared to writing the for loop with indexes.

Coming back to loops and arrays

Remember the example before on why array are more useful than normal, single variables?

It might be unclear on how to *populate* an array with user data, so here is a simple method on asking the user for every single input to the array

```
arr = []

while True:
    user_input = input()
```

```
if user_input = "":
    break
arr.append(user_input)
```

Try this and give it some data and probably print the array after the user has given the data to the array.

Challenges!

Challenge no. 1

Make a Python program that that asks the user to input any number of numbers from the user and then calculates the sum, the average, and the range of input. As an additional requirement, make sure that the user is actually inputting numbers instead of nonsense.

Challenge no. 2

Take 10 numbers from the user into an array and then: 1. Print the first number given 2. Print the last number given 3. Print the array, but without the first element nor the last 4. Print a sorted version of the array

Challenge no. 3

A classic: "fizzbuzz"

Iterate through the numbers from 1 to 100 and do the following:

1. Print "fizz" if the number is divisible by 3
2. Print "buzz" if the number is divisible by 5
3. Print "fizzbuzz" if the number is divisible by 3 AND 5

Hint: remember the modulus operator: %

Lecture 4 Notes

March 6, 2020

Lecture 4: Functions and Object Oriented Programming

Today we will cover two of the most important concepts in programming which are functions and objects.

Functions

Functions are named sections of a program that performs a specific task. In this sense, a function is a type of procedure or routine. This routine performs a tasks and in some cases returns a value. This is how you create a function in python:

```
def hello():
    print("Hello buddy!")
```

The keyword `def` signifies the definition of a function, then follows the name of the function, in this case `hello`. Then the parenthesis, which we will discuss after too, and at the end `:` which always signifies the end of a statement in Python.

To call the function we created we simply use:

```
hello()
```

The parenthesis contain the **arguments** which are parsed in the function. Arguments are variables which you want the function to work with. A simple example of this is the function `print` which we have covered already:

```
print("Hello buddy!")
```

`print` is the name of the function, and the string `"Hello Buddy"` is the argument parsed into the function.

Here is an example of how arguments could be used in your own functions:

```
def addition(x, y):
    print(x+y)

addition(1, 2)
```

Here we are parsing the two integers `1` and `2`, the function `addition` is taking them and printing the sum. Now we will cover the return of a function.

Return

A return is a value that a function returns to the calling script or function when it completes its task. You will be using this a lot if you keep programming. Let's look at and example:

```
def operation(x, y):
    result = (x+y)*2
    return result

r = opeation(3, 6)
print(r)
```

We are parsing the integers `3` and `6` into the function `operation`, the variable `result` which is defined within the function, is returned. This means that the variable `r` will have the value of whatever is returned from `operation`

There is also the possibility to nest functions, which is a very neat way to clean up your code. Don't overdo it or it will look like crap though! :)

```
def operation(x, y):
    result = (x+y)*2
    return result

print(opeation(3, 6))
```

It is important to know that once a function returns, nothing else within that function will execute. (*yield*)

When would you use this?

Functions are great to split up your program into parts and run the same operations multiple times without having to re-write it.

Objects

Another concept implemented in programming languages like python, is Object Oriented Programming (OOP). Objects can contain data, in the form of fields (often known as attributes or properties), and code, in the form of procedures which we can call functions or methods.

```
class Dog:
    def __init__(self, name, age, breed):
        self.name = name
        self.age = age
        self.breed = breed

    def bark(self):
        print("Bau Bau")

    def sit(self):
        print(self.name+" sits")

    def talk(self, message):
        print(self.name+" says: "+message)

    def __str__(self):
        return self.name+":"+self.breed+":"+str(self.age)
```

We define the object by using the syntax `class` followed by the name of the object, in this case `Dog` and then the `:` since this is a statement.

The Constructor

The constructor of an object is a subroutine which creates the object. In python we define the constructor of the object using the syntax `def __init__(self):`. **self** is a special keyword which represents the object itself. It is used to set fields (attributes) of the object. The keyword `self` must be parsed to the methods of the object which require access to the object's fields.

In our examples we are setting the fields `self.name`, `self.age` and `self.breed` which will be avaiable to all the methods using the keyword `self`.

Other than `self` we can parse other arguments in the constructor, usually they are used to set the fields of the object.

Methods

Methods are just like functions, and you define them inside the object. These methods can take the arguments `self` but also any other argument you want your method to use. An example using the `Dog` object is:

```
def talk(self, message):
    print(self.name+" says: "+message)
```

We are accessing the field `name` and printing it along with the message parsed in the method.

Using the object

Now we have defined an object, it's time to use it.

```
d = Dog("NotAPlanet", 6, "French Bulldog")
print(d.name)
print(d.breed)
print(d.age)
d.bark()
d.talk("Hello, I'm not supposed to talk")
d.sit()
```

As you can see we simply create our object by calling its name `Dog` with brackets and inside the arguments to parse to the constructor. The variable `d` will now be an object of "type" `Dog`. We can access the object's fields and methods just by calling the object name `.` field/method.

Representations

Representations are really fun. They allow you to assign a type representation to an object. In our case it is the `string` type representation, which is defined using the name `__str__` as the name of the method. We can get the "stringfied-dog" using:

```
print(str(d))
```

We can use this to create more complex programs, for example if we had an array of dogs, we could set the `int` type representation to be their age using `__int__`. This would allow us to for example sort the dogs by their age.

Recap

So today we covered a lot of stuff so here is a little summary:

- Functions:
  - Arguments: variables given to the function to work with
  - Return: values returned to the function call when all operations are finished
  - E.g. "'python def operation(x, y): result = (x+y)*2 return result r = opeation(3, 6) print(r) "'

That' all there is to functions.

- Objects:
  - Constructor: creates the object and initially sets the object's fields
  - Methods: functions exclusive to the object
  - Self: the object itself, which is used within to call fields or methods
  - Representations: the type representation of an object
  - E.g.

```
class Dog:
    def __init__(self, name, age, breed):
        self.name = name
        self.age = age
        self.breed = breed
```

```
def bark(self):
    print("Bau Bau")

def sit(self):
    print(self.name+" sits")

def talk(self, message):
    print(self.name+" says: "+message)

def __str__(self):
    return self.name+":"+self.breed+":"+str(self.age)
```

Challenges

1) Create a Person object which has at least the following fields:

   - name
   - age (int)
   - job
   - motto

   Then have methods which allow the Person object to introduce himself and add a string represenation of the object which returns the motto.

2) Create an array of Person object, then have a function which takes the array as an argument and returns the oldest Person in the array. (You will have to use loops to find it out)

Review and Challenges!

March 6, 2020

What we will be doing

A load of challenges and some review

If there is something you want clarification on or some questions: ask us now!

Challenge no. 1

Write a program that plays "rock, paper, scissors" with the user. Also implement checking for a win and output who won.

Challenge no. 2

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23.

Find the sum of all the multiples of 3 or 5 below 1000.

(Taken from Project Euler)

Challenge no. 3

You are given a file with 'masses' on each row. Lets imagine that these masses are a part of a rocket that is about to fly and we need to figure our how much "fuel" we need to launch the rocket. The equation to get how much fuel we need for a mass is

$$\text{floor}(\frac{m}{3}) - 2$$

What is the sum of the fuel needed to launch the masses given in the file?

See the github repo for the input file.

(Taken from Advent Of Code 2019)

Challenge no. 4

The sum of the squares of the first ten natural numbers is,

$$1^2 + 2^2 + ... + 10^2 = 385$$

The square of the sum of the first ten natural numbers is,

$$(1 + 2 + ... + 10)^2 = 55^2 = 3025$$

Hence the difference between the sum of the squares of the first ten natural numbers and the square of the sum is 3025 - 385 = 2640.

Find the difference between the sum of the squares of the first one hundred natural numbers and the square of the sum.

Challenge no. 5

Given an array of some lenght with numbers, sort that array.

Answers

Here are the answers so you can check if your program runs correctly

Challenge 2: 233168

Challenge 3: 3464458

Challenge 4: 25164150

Function and Object Exercises

March 6, 2020

What we will be doing today

Review of functions and objects alongside exercises

Any questions?

Function Problems:

1. Make a function that takes any number array and returns the product of all the integers in that array
2. Make a function that returns the average of the array
3. Make a function that deletes the last element of an array and returns the modified array.
4. Make a function that takes two arguments: weight in kg and height in cm. The function should calculate the BMI from that and return it.
5. Make a function that takes an array and a number (n). It should return the n first elements of the function. Beware of n longer than the array

Object problems

1. Point object
    1. Make an object that is a point with the coordinates x and y so that i can create a point by calling

    ```
    p = Point(2,3)
    ```

    such that 2 is the x coordinate
    2. Make a function that takes two points and returns the distance between them
2. Make an object called Human that holds qualities such as height, name, weight, and age. Make the constructor for that

Libraries and Sockets

March 6, 2020

What we will cover today

- Libraries? What are they and how we use them in Python
- Exploring the socket library
- A cool project with socket

Libraries

A library is a collection of code that can contain functions, objects and variables which we wish to use in different programs. Libraries make our life easier because it's code we can use in our programs quickly without having to "re-inventing the wheel".

Example of library

A very common example of a library is the library `math` which contains a lot of helper functions to do math.

```
import math

print(math.sqrt(2))
```

To import a library we use the keywork `import` followed with the name of the library we want to use. Once a library is imported we can use it by calling the name of the library `.` whatever function/variable/object we want.

Selective import

Some libraries are very big and sometimes we only wish to import a certain thing from them. We can import a specific thing from a library without loading the whole thing like this:

```
from math import sqrt

print(sqrt(2))
```

We use the keyword `from` followed by the name of the library, then `import` and whatever we want to import from the library. In this case we import the function `sqrt`, which can be used by simply calling it. If we want to import multiple things from a library we simply separate them with `,` or if we want to import everything without having to use the name of the library every time we can use this : `from math import *`

Renaming libraries

If we are feeling very lazy we can rename libraries so we don't have to use the name everytime we want to use something:

```
import math as m

print(m.sqrt(2))
```

We use the keyword `as`.

Socket library

The socket library is a library used for anything regarding low-level networking with python. This is how we import it:

```
import socket
```

Today we will be creating a simple client-server application which allows us to send data to each others when we are on the same network

Working with interactive Python

To play around with socket we will use the python3 interactive shell which you can access by running `python3` on your terminal. On this shell you can write and run python code quickly without having to create a script.

First of all choose a friend to work with and find out each others local ip address.

The server

```
>>> import socket
>>> s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>>> s.bind(("0.0.0.0", 4444))
>>> s.listen()
```

```
>>> conn, addr = s.accept()
>>> print("Connected with", addr)
>>> data = conn.recv(1024)
>>> print("Received:", str(data))
>>> conn.sendall(b'Your reply')
```

The client

```
>>> import socket
>>> s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>>> s.connect(("friend.ip.0.0", 4444))
>>> s.sendall(b'Your message')
>>> data = s.recv(1024)
>>> print("Received:", str(data))
```

Challenge

Your challenge for today is to create a more interactive program to communicate with your friend. Right now we can only send one fixed message and one fixed reply, how can we improve this?

Lecture 8 Dictionaries

March 6, 2020

What is a dictionary?

A Dictionary cointains **keys** and **values** that the keys correspond to.

How do you use a dictionary?

Dictionaries work similarly to arrays in that sense that you can access it with '[]'.

```
# Declaring a dictionary

adict = {
    'a': 1,
    'b': 2,
    'c': 3,
}

adict['a']      # => 1
adict['b']      # => 2
adict['c']      # => 3
```

Different dictionarys

Different types of keys:

```
specialdict = {
1: 2,
'b': True,
'abba': 'baab',
}

specialdict[1]      # => 2
specialdict['b']      # => True
specialdict['abba']      # => 'baab'
```

Iterating over a dictionary

Using the 'adict' we declared before, we can iterate the dictionary by:

```
for key, val in adict.items():
    print(key, ":", val)
```

This will output `a : 1     b : 2     c : 3`

An example: Phonebook

We start by declaring a phonebook dictionary that maps each of the stored names to a phone number:

```
phonebook = {
    "james":  2025550112,
    "anna": 2025550187,
    "jacob": 2025550196,
}
```

With this we can return the phone number of James by typing

```
phonebook["james"]
```

Adding to dictionaries

The amazing thing that we can do with dictionaries is that we can add to them values in a dynamic way

For example, to add a phone number for new person called Mary for example, we would just type:

```
phonebook["mary"] = 2025550131
```

And we can do this as many times as long as we have unique names

If we want to alter a phone number, for example James has changed his phone number, we type:

```
phonebook["james"] = 2025550129
```

Deleting from dictionaries

In order to delete from dictionaries we can type

```
del phonebook["anna"]
```

In this case we have deleted the person Anna and her phone number from the phonebook completely.

Checking and accessing

If we get an user input of a key (uinput) in a dictionary (adict), we first have to check if that key exists in the dictionary.

```
uinput = input()
if uinput in adict:
    print(adict[uinput])
else:
    # Do something else
```

This will print the value at the key uinput if it exists in adict and does something else otherwise.

Excercises

1. Write a phone-book of your own with three people to start with. Then add one, modify one and delete one from the phonebook
2. Try and print a dictionary as it is without converting it to a string. What do you see?
3. Write a library dictionary. The key should be the name of the book and each value should be ANOTHER dictionary that contains name, page count, etc. Add at least three books and print one to the terminal.