200 PYTHON EXERCISES

An Introduction to Python

FRANK ANEMAET

200 Python Exercises: An Introduction to Python

Frank Anemaet

This book is for sale at http://leanpub.com/learnpythonbyexercises

This version was published on 2020-10-18

Contents

Chapter 1: Python

Start with Python, run that code!

This chapter will be about getting started with Python. We will begin by explaining how to run Python code online, then move on to how to get Python running on your system. At the end of this chapter, you should be able to run Python code.

Run Python Online

If you go to the website https://repl.it/languages/Python3 you can run Python 3 in an online interface. So on the leftside you'll see you can have more than one file. By default is shows the main.py file. Every

Python script ends with the .py extension.

Then you have an editor here where you can type your code. This is where you write all your Python code.

Then on the right side you have a terminal, which shows the output of your program.

Python programs don't run by themselves, you have to click the run button. That's the big green button above the code.

Some other sites to run Python code online are:

- https://www.onlinegdb.com/online_python_compiler
- https://rextester.com/l/python3_online_compiler
- https://skulpt.org/

First programs

Python programs are executed top down. It starts at the the top of the file, then steps down. In other words, it's executed line by line. Every line is called an instruction or statement. Every Python file ends with .py, just like pdf files have a .pdf extension.

Unlike other programming languages that require many lines for simple programs, Python code can be quite compact. You can try this (one line) program:

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

Click run and you'll see the output on the right. That's right, it shows the message "Hello world".

That's it, you made your first program!

Anything you type between the quotes will be shown on the screen. That can be a word, a sentence, a paragraph or a whole book. You can use a double quote " or a single quote '.

Wondering why the function is named print?

It's named print for historic reasons. Back when programming languages were first designed, there were no screens and program output was printed on paper.

You can show multiple messages on the screen. To do so, change the program into this:

```python
print("Hello World")
print("I am learning Python")
print("Bye bye")
```

This works for variables too. For example, you can set a variable to 3 and output that variable.

```python
x = 3
print(x)
```

You can display a variable as many times as you want.

```python
x = 3
print(x)
print(x)
```

Variables can change value during the program.

```python
x = 3
print(x)
x = 4
print(x)
```

That's why we see 3 and 4 in the output.

So this is how Python programs are executed, top down. You've seen how to use the print function, which can output text and variables.

Fact: Python programs are always executed top down. It may jump around using function calls or loops, but the order is top down.

If you want to add an extra newline, you can use the special character backslash `\n`. Like this:

```python
print("Hello\n")
print("World\n")
```

The backslash n character (it shows as two characters, but it's a single newline character) creates a new line.

The are other special character like which displays a tab and ' which displays a quote.

Run Python locally

Notice: Running locally is optional, all code can be executed online.

On many Linux systems, Python is installed by default. On Windows or Mac you need to install it. This is only if you want to run Python locally.

If you want to install Python locally, go to https://www.python.org/
Then download the version for Windows or Mac and install it. Run the installer.

Then in the command line you can type "python" to open the Python interpreter.

To see which version you have installed type

```
python --version
```

It's recommended to use Python 3.0 or newer.

To run a Python script type

```
python example.py
```

If you run Python locally, you should add this on top of the file:

```
#!/usr/bin/python3
```

This lets your computer know where Python is located.

You can write your python programs in any text editor you want (yes even notepad). But I recommend going with an IDE (see below) because it makes softare development much easier.

IDE

You do not have to code from the command line. There are intergrated desktop environments (IDEs). These are tools software developers use when making software.

Some popular IDEs are:

- PyCharm
- Visual Studio Code
- Spyder IDE
- Wing Personal

An IDE lets you start a Python project, has syntax highlighting, numbering, tabs, debugging and many other features that make your life as software developer much easier.

Summary

You should be able to run Python code. Python can be run both online and locally (on your system).

Every Python file ends with the .py extension. Programs are executed top-down and you can output text or variables with the print function.

Exercises

1. Can you run Python online?
2. Can you run Python offline? (on your computer)
3. What is the most recent version of Python?
4. How can you run Python?
5. Which (text) editors can be used to create Python programs?
6. What file extension do Python programs have?
7. Create the "Hello World" program, output "Hello World"
8. Create a program that outputs your name. Make it output your last name on the 2nd line
9. What is the difference between "Hello world" and 'Hello World' ?
10. Create a program that prints a christmas tree.
11. Create a program that prints a rectangle.
12. Create a program that prints a house (in ASCII art style).
13. Create a program that prints your name using the '*' character
14. Define a variable age, and try to output "My age is … years old"
15. Define a variable name with a value (e.g. Tiffany), and output "Hello Tiffany"
16. Make a program that outputs different currency symbols (dollar, euro etc)
17. Try to output the non-English character epsilon (ε)
18. True or False: Python has only existed from year 2010

19. Can Python programs be run without having Python installed?
20. What file extension is commonly used for Python programs?
21. What is the advantage of Python over C++?
22. What is the advantage of Python over Java?
23. Create a program that calculates `52 + 48`
24. Create a program that calculates `30-10`
25. Why use an IDE over an text editor like notepad?

Chapter 2: Variables

Variables

This chapter will be about variables with Python. We will begin by explaining the basics. At the end of this chapter, you should be able use variables in Python.

Define variables

You can easily define variables. Let's say a variable x which has some number.

```python
x = 7
```

So this is a variable named x. You can have multiple variables in your program.

```python
x = 7
y = 3
```

You can output them variable x and y printed to the screen.

```python
print(x)
print(y)
```

Now the variable doesn't have to be a single character. You can have a variable, say name, with the value of your name. Variables can have text values (also called strings) or numeric values.

```python
name = "Alice"
print(name)
```

So you see variables can be one character along with two can also be put in one character along so we'll see.

Variable naming

In the last paragraph you have defined a few variables like x, y and name. Can any variable name be chosen?

No, there are some rules for naming variables in Python. Many things are allowed by not everything.

The rules are:

- A variable name must start with a letter or an underscore character
- It can only contain alpha-numeric characters and underscores
- Variable names are case-sensitive (var, Var, and VAR are 3 different variables)
- Can't start with a number

Variables can contain capital letters. So instead of having lower case it can be upper case as well.

In Python programming code you often see shorthand variables (x,y,z) or very specific variable names (lightSwitchOne, serverInAmsterdam). The later style is named "Camel Case".

Table 0.1: Demonstration of allowed variable names

Name	**Allowed**
Name	Allowed
x	OK
y	OK
name	OK
firstName	OK
first_name	OK
~x	Wrong
#n	Wrong
X	OK
T	OK
2x	Wrong
_x	OK
mX	OK

Variables in operations

Variables you be used in operations.

Let's define variable z, which is an expression x and y. You can output that to the screen.

```
x = 3
y = 4
z = x + y
print(z)
```

All the operations work, division (/), multiplication (*), substraction (-) and addition (+).

So you can do different operations with variables.

Conversion

Variables have a data type. Just like not all animals are cats, not all variables are text variables. There are different types of variables.

If you have a numeric variable, you can convert it to a text variable. What's the difference you say?

In computing 2 is not equal to "2". One is a number, the other is text (string).

Text in the computer is made using an ASCII table, or for modern apps, UTF-8 table. This is a character table with many dierent characters. For UTF-8 you can find many foreign characters too.

To convert a number to a string, all you have to do is call the str() function with the variable as parameter.

```
x = 3
print("x is " + str(x))
```

That outputs x as a text value.

How can you reverse text (string) to number?

It depends on what kind of number you want. For whole numbers you can use integers, for other numbers use floats.

```
x = "3"
x = int(x)
z = x * x
```

Summary

You should be able to use variables in Python code. Variables have a data type (string, numbers) and can be converted. Operations can be applied on variables.

In the next chapter we'll discuss strings (text) in detail.

Exercises

1. Is variable x and X the same?
2. What is the output of this program?

```python
x = 3
print(x)
x = 4
print(x)
```

3. What is the difference between `x = 3` and `x = 3.0` ?
4. Is $y an valid variable name?
5. Is _y an valid variable name?
6. Is %y a valid variable name?
7. Is ~z a valid variable name?
8. How can you output "the variable x contains 3" if x=3 ?
9. What does `int(x)` do?
10. Is myCar a valid variable name?
11. Is MyCar a valid variable name?
12. Is my_car a valid variable name?
13. Create a program that calculates the VAT on a price, use variable revenue and variable total.
14. Create a program that calculates the body mass index.

    BMI = (Weight in Kilograms / (Height in Meters x Height in Meters))

For Brits,

BMI = (Weight in Pounds / (Height in inches x Height in inches)) x 703

15. Write a program that converts kilometers to miles. 1km is equal to 0.62 miles
16. (based on 15) now write a program that converts miles to kilometers
17. Given variable x = 4, write a program that calculates the square, cube of x
18. What happens if you divide by zero in Python?
19. Does Python support negative numbers?
20. What is the output of 7/2?
21. Can you define multiple variables on one line?
22. Make a program that exchanges the values of variable x and y, where x = 2 and y = 5. It should become y = 5 and x = 2
23. Define `x = 5` then call the function `type()`. Now change it to `x = 5.0` and call it again. Why does the output differ?
24. If you start the Python interpreter, you immediately see the output (after >>>). Why would you store your program in a *.py* file?

```
Python 3.8.2 (default, Jul 16 2020, 14:00:26)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> x = 1.9
>>> y = 0.1
>>> x + y
2.0
```

25. Once you defined a variable, can you change its value again?

Chapter 3: Strings

What are strings and how to use them. This chapter will be about strings. You can think of a string as text. Almost every computer program uses strings, because it either needs to output text to user or save text. In other words, strings are a
core part of almost any computer program.

You can define variable named s which has the text Hello. For instance, you can do it like this:

```
s = "Hello"
print(s)
```

So variable s in this case is a string, because it has a text. The text can be as long as you want, it can be a letter, a word, a sentence, a paragraph or even a whole book.

How do you add multiple sentences into a string? You can't press the enter key a few times, instead you use the special character \n.

```
s = "Hello\nWorld\nHello"
print(s)
```

So you can have multiple lines in s even though it's a single string.

Concatenate

You can also concatenate strings. Concatenation is adding them together.
For instane, in the example below we add three individual strings together.

```
s = "Hello" + " " + "world"
print(s)
```

If you run this program you will see the message "Hello world". Thus the + operator can be used for concatenation.

This can be done for variables too:

```
s1 = "Hello"
s2 = "world"
print(s1 + " " + s2)
```

This will concatenate the string variables s1 and s2.

String replace

String variables come with methods. You can use these methods to change the text.
Lets start with a simple string variable:

```
s = "Hello world"
print(s)
```

What if you want to change "Hello" into something else?
You can use the string variables replace function.

```
s = "Hello world"
s = s.replace("Hello", "Goodday")
print(s)
```

So you see, every string objects has a .replace() function. The first argument is the source text, the second argument is what to replace it with. Lets try that again:

```
s = "Hello world"
s = s.replace("world","Haley")
print(s)
```

This will replace world with the name. You can replace with a single character, a word or a sentence.

Replace special characters

You can also replace spaces, commas or other special characters. To replace spaces with underscores, you can use this:

```
s = "Hello world"
s = s.replace(" ","_")
print(s)
```

You can replace newlines (\n) or tabs (\t) too,

```python
s = "Hello Alice.\nHow are you doing?"
s = s.replace("\n",",")
print(s)
```

User Input

Let’s make our first interactive program. We are going to create a program that not only outputs text but also gets user input (keyboard input).

The computer program will ask your name and put it on the screen. If you use a very old version of Python, like Python 2, you need to use raw_input() instead.

```python
name = input("Enter your name: ")
print(name)
```

Run it, you can type your name and it puts the name to the screen. You can get a lot of information like this. So let’s get some information.

```python
firstName = input("Enter name: ")
lastName = input("Enter lastname: ")
country = input("Enter country: ")
print(firstName + " " + lastName + " " + country)
```

Then you can output all 3 variables we got with input methods. OK run it. You can enter information and then it outputs that to the screen. This works for sentences too, not only words.

Summary

You should be able to use strings in Python. You can define strings, concatenate strings and replace its content. You can make interactive programs, by asking the user for keyboard input.

Exercises

1. What is a string?
2. What is the minimum length of a string?
3. Create a program that asks for your name and outputs it
4. Create a program that asks for your name, address, city and phone number
5. Combine the characters ‘a’,’p’,’p’,’l’,’e’ into a string by concatenation
6. What does \n do?
7. What does \t do?

8. On the internet you can sometimes see *raw_input()* instead of *input()*, what is the difference?
9. Why does *input()* have brackets?
10. Given the string "Hello World", replace the word "World" with "Python"
11. Can you call *print("Hello World")* without brackets? why?
12. Create a program that adds the word "good" to the words "morning", "afternoon" and "evening" such that the output is:

```
good morning
good afternoon
good evening
```

13. Why does this program throw an error:

```
s = `hello world`
print(s)
```

14. What is the output of this program:

```
x = "hello"
y = "world"
print(x + y)
```

15. What is the output of this program:

```
1x = 'kitchen'
y = 'tuple'
print(1x + y)
```

16. What is the output of this program:

```
print(3 * "hello")
```

17. Is this program valid?

```
s = 5 * 'a'
y = 3 * 'b'
s + y
```

18. Is this a valid program?

```python
x = "hello".replace("e","o")
```

19. Is this a valid program?

```python
x = "hello".replace("e","o").replace("h","g")
```

21. How can you output the phrase “What’s up?”
22. How can you output “the price is $5.50”?
23. What is the output of `print("hello\\n")` ?
24. What is the output of `print("a\tb\tc\n5\t6\t7\n")` ?
25. What is the output of this program?

```python
>>> x = "the world"
>>> y = "is not enough"
>>> x = x + " " + y
>>> x + y
```

Chapter 4: Lists, a collection in Python

This chapter will be about lists, Python has support for lists. A list is a collection. Just like real life lists (shopping lists, todo lists), a list can contain zero or more items. You can define a lists, let's say shopping list.

```
shoppingList = [ "Apple", "Bread", "Eggs", "Cookies" ]
```

You have to use these cubic brackets when creating a list. That's how Python knows it's the lists. The list can contain several items and can be of dierent data types (numbers, strings)

```
myList = [ 1, "Alice", "Car", 3912 ]
```

Access list items

To print a specific item you can use the index. This must be in the special brackets []. Unlike humans, computers start counting from zero and Python does too.

```
myList = [ 1, "Alice", "Car", 3912 ]
print(myList[0])
```

Zero is the first element of the list. In this case number one. By using the zeros index, it'll show you the first item of the list.

So what would be the second item of the list? That's right, number one.

```
myList = [ 1, "Alice", "Car", 3912 ]
print(myList[1])
```

Number one (the second element), would be Alice. This goes on, so two will be the third item. So with any list, index zero is the first item.

What if you want to have the last item, and you don't know how long the list is? Then a minus 1 will start counting backwards, regardless of how long the list is.

```
myList = [ "Car", "Plane", "Boat", "Bike" ]
print(myList[-1])
```

So even if you have a list of 200 or 300 items or longer, you can immediately get the last item. The minus character makes it count backwards.
You can change the values of list items. Again, you use the index.

```
myList = [ "Car", "Plane", "Boat", "Bike" ]
myList[0] = "Rocket"
print(myList[0])
```

Delete items

You can also delete items from a list. So if you create a shopping list, you have to specify the index. For a list of 3 items, thats either zero, one or two.
This will remove the first item.

```
items = [ "Eggs", "Milk", "Bread" ]
del items[0]
```

To remove the second item, use index one.

```
items = [ "Eggs", "Milk", "Bread" ]
del items[1]
```

You'll see the second item has been removed, but the first and last item are still there. So that's how you can remove elements from a list.

Append to list

You can also append to a list. To do that, write .append aer the list name. You can add one or more items to your list.

```
items.append("Fish")
```

And then if you run it's got a longer list now because it's happened as these two items to the list. This will add the new elements to the end of the list.

List magic

If you create a list, you can immediately get to maximum in the lists. This is very simple to do. First define a list, say a list of numbers

```python
grades = [ 61,56,42,67,73,12 ]
print(max(grades))
print(min(grades))
```

If you run it you will get the maximum and minimum number. You can also get a length of a list and the length, no need to count the number of items manually.

```python
grades = [ 61,56,42,67,73,12 ]
print(len(grades))
```

You can save these values if you want. If you want to use them later in your program. To do that, you can do this:

```python
grades = [ 61,56,42,67,73,12 ]
minimum = max(grades)
maximum = max(grades)
length = len(grades)

print(minimum)
print(maximum)
print(length)
```

Be careful with the opening and closing braces, they should always match.

Summary

Lists are collections. They can contain zero or more elements of dierent data types or the same data type. You can access a list element using an numeric index, starting from zero. If you use a negative index it will count backwards. After creation of a list you can still add (append) and remove (del) list elements. Statistics can be calculated from numeric lists.

Exercises

1. What is a list?
2. Can a list contain only numbers?
3. What's wrong with this program?

```
a = (1,2,3)
```

4. Create a list of numbers
5. Print the maximum value in the list
6. Print the minimum value in the list
7. Now output the length of the list

8. Create a list with the names of your friends
9. Define a list of drinks and print the first drink in the list
10. Print the length of the previous list
11. Output the last item in the list

12. Create a list fruits and add “apple” to the list
13. Now add the fruits “melon” and “lemon”
14. Can you add two lists together?
15. Delete the first item from the list
16. What’s wrong with this code?

```
x = [1,2,3]
print(x(0))
```

17. What’s wrong with this code?

```
x = [1,2,3,,]
print(x)
```

18. Is the program below correct?

```
print([1,2,3,4])
```

19. Is this program correct?

```
print([1,2,3] + [4,5,6])
```

20. Are x and y the same?

```
x = [1]
y = 1
```

21. Does this program work?

```python
x = []
del x[0]
```

22. What is the output of this program?

```python
x = [1,2,3]
print(x[-1])
```

23. Explain the program below:

```python
x = [[1,2,3],[1,2,3]]
```

24. What does this program output?

```python
x = [[1,2,3],[1,2,3]]
print(x[1][0])
```

25. What is the output of this program?

```python
x = [[5.0, 3.5, 2.0], [1.0,2.0,1.0], [4.5, 3.5, 5.4]]
print(x[0][0])
print(x[1][0])
print(x[2][0])
```

Chapter 5: Tuples and dictonaries

This chapter will be about tuples and dictionaries. You have already see a type of collection (lists), a tuple and dictionary is another form of collection. Personally I seldomly use tuples, but it's good to know what they are and how to use them. Dictionaries are a very useful form of collection.

Another data type of variable is called tuple. Its like a list with the main dierence that cannot be changed once created.

So let's create a tuple and you'll see we use a dierent type of brackets here.

```python
months = ("January", "February", "March", "April")
print(months)
```

So in a list you will have data that changes but in a tuple that cannot be changed. If the data should never change, use a tuple. Otherwise use a list.

Tuple objects cannot change the values. That's the main difference which tuples and lists. Now if you have a single elements you'll have to define it like this

```python
months = ("January",)
print(months[0])
```

You can use the index to get an element, just like with lists. If you have an empty list would just be these brackets.

```python
months = ()
```

Want to change anyway?

A tuple cannot be modified, you cannot delete items or append items. The data stays as it is.

What if you want to change the data?

There's a trick. You can convert a tuple to a list and then to a tuple again.

```python
months = ("January","February")
months = list(months)
months.append("March")
months = tuple(months)
```

Dictionaries

Python supports so-called dictionaries. Just like a real world dictionary, for every word there is a value. So we can have a dictionary of words.

Dictionaries use another type of brackets than lists or tuples.

```
1 d = { }
```

So just by typing these, Python knows that we're going to make a dictionary.
So as I said there's a key-value mapping. So for example

```
words = { "A": "Apple", "B": "Bee" }
print(words["A"])
```

Then we can use the index. In this case "A" is the key and "Apple" is the value. These form a pair. The next pair, "B" is the key and "Bee" is the value. When you run this it will output the value for key "A". So that's how these key and values are mapped.

More dictionaries

Now you can create any kind of dictionary. Instead of a single character you can map more complicated character. So let's say we are mapping English to Spanish, then you can use the keys.

```
words = { "Book": "Libro", "House": "Casa" }
print(words["Book"])
```

The key can be a single character ("A","B") but it can also be a word ("Book","House"). Again, there's a key value mapping here. Every key is mapped to a single value.

Dictionary keys and values

You can have a big dictionary. Then you can get all of the keys of the dictionary at once:

```
print(words.keys())
```

Likewise you can also get the values in a dictionary

```
print(words.values())
```

so that will fetch the values and you'll see the values for a dictionary.

Appending

How do you add new new value?

You can define the dictionary name, the key and then what value to map.

```python
words["Good"] = "Bien"
```

Likewise you can delete values.

```python
del words["Book"]
```

You'll see the first key value pair has been deleted. Now one thing to know is that these dictionaries can not only contain text but also values. So you can have another type of dictionary in which you map for example the numbers.

```python
numbers = { "One": 1, "Two": 2, "Seven": 7 }
print(numbers["Seven"])
```

So dictionaries can not only contain text or strings but also numbers.
A dictionary does not keep order, the order of the dictionary can change.

Exercises

1. Define a tuple with some items
2. What is the difference between a tuple and list?
3. Try to convert a tuple to a list
4. Convert a list to a tuple
5. Print items in a tuple and list
6. What is a dictionary?
7. What is the purpose of a key in a dictionary?
8. Is a dictionary ordered?
9. Is a tuple ordered?
10. Is a list ordered?
11. How do you output the values in a dictionary?
12. How do you output the keys in a dictionary?
13. Create a list and delete an item
14. Create a tuple and delete an item
15. Create a dictionary and delete items
16. Is a list mutable?
17. Is a tuple mutable?
18. Is a dictionary mutable?
19. Output a dictionary as a table
20. Can a dictionary contain both numbers and strings?
21. Create a program where you define a list, a tuple and a dictionary. How do they differ?
22. Can a list contain other datatypes than numbers?
23. Create a program that asks for names and adds them to a list of names
24. Create a program where the user can define the dictionary items
25. Define a dictionary and output the keys and values

Chapter 6 Loops

This chapter will be about iteration (repetition). In Python you can repeat one or more lines (codeblock) using loops. Python supports for loops and while loop, which work differently but both repeat code.

For loop

You can repeat certain times. So if you have a collection, you can loop over each items. Keep in mind Python indention, you need four spaces.

```
names = ["Alice","Alex","Bob","Brittney"]
for name in names:
    print(name)
```

When you run it, you see each item in the list. This works for any kind of list.

Range function

The range function is special, because you can do this for numbers too! This saves you a lot of time, instead of manually doing this.

You can use the range function to create a list. Say X is a list of the range 1 to 11 then, prints x and you'll see we now have a list of numbers 1 to 10.

```
x = list(range(1,11))
print(x)
```

If you were to put 101 we'd have a list of numbers 1 to 100. You can also use that in a loop.

```
start = 1
stop = 10
step = 1
for i in range(start, stop, step):
    print(i)
```

This saves you a lot of time, instead of manually doing this. This will count over the numbers with the step size of one. You can change the starting, stopping and step size numbers.

You can use the range function when you want to deal with numbers in a loop. If you want to use strings or lists of strings, in general you don't use the range function.

While Loop

There's another kind of loop: the while loop. This loop can repeat indefinitely or until a condition is true. For example, if you have a loop like this it will never increase

```
i = 1
while i < 10:
    print(i)
```

This is an infinite loop. What you want to do is to increase i inside loop.

```
i = 1
while i < 10:
    print(i)
    i = i + 1
```

That way, the variable i increase and eventually i is larger or equal to ten. This type of loop is very useful when you want to wait for something to happen, like waiting for keyboard input, mouse input or network traffic.

While else

Else can be used in a while loop. This is not so common, but it's possible.

```
i = 1
while i < 12:
    print(i)
    i = i + 1
else:
    print("i greater than 12")
```

See it repeats the block of code, then it'll go into this block of code when it's finished. In this way, when the loop is finished you can run another block of code.

While input

You can get keyboard input in a while loop. For instance, while x is not 10 keep asking the number.

```python
i = 1
while i != 6:
    i = int(input("Enter i: "))
    print(i)
```

That will repeat the loop until the value of i is guessed. This works for a text user interface too:

```python
x = input("Enter command: ")
while x != "stop":
    print("Unknown command, try again")
    x = input("Enter command: ")
```

This code block will repeat until finally the command “stop” is typed.

Exercises

1. What is the purpose of a for loop?
2. When does a for loop end?
3. Can for loops only be used for output?
4. Can a for loop be inside a for loop?
5. What is indention?
6. Why is indention necessary to use in for loops?
7. Can a for loop use a list?
8. Can a for loop use a tuple?
9. Can a for loop use a dictionary?
10. Can you use two for loops for a two dimensional list?
11. What does the range() function do?
12. What does a while loop do?
13. Use a for loop to show the items in a list
14. Use a while loop to show the items in a list
15. What is the meaning of != ?
16. Can you do two comparisons in a while loop?
17. Can you use a for loop on a string?
18. Create a list of even numbers in a for loop
19. Create a list of odd numbers in a for loop
20. Create a program that asks for commands, if the user types ‘quit’ it should stop
21. Create a loop that outputs the numbers 1 to 100
22. Create a loop that outputs the multiplication table of 5
23. Which comparison operators can be used in a for loop?
24. How many for loops can be inside a for loop?
25. How many while loops can be inside a while loop?

Chapter 7

If statements

This chapter will be about if statements. If statements are used for selection, should it run this code or not. In real life you use if-statements too. For instance, "if shoes on, then go outside". By the end of this chapter, you'll be able to use if statements in your own programs.

If statements run code only if the variable meets a condition. For instance, only if x is tree, execute code.

```
x = 3
print('Begin')
if x == 3:
print('Do this')
print('End')
```

For equality check you need (==), for greater than (>), greater or equal (>=), smaller than (<), smaller or equal (<=) and for unequality (!=).

This will print "Do this" only if x has the value of 3. If you change the value of x you'll see the code is not executed. The code block within an if statement needs four spaces. Let's say you have a variable y and then y is greater then 10 do something.

```
y = 50
if y > 10:
    print('begin of codeblock')
    print('do something')
    print('end of codeblock')
```

So you see it runs the code but only if y is greater than 10. So if I would put y=1 here, nothing is shown on the screen.

So that's what this statement does: It only executes a block of code if condition (y > 10) is true. So you can have multiple lines, a codeblock, or a single line of code.

Combined if statements

If statements can also be combined. Let's say we have X, if x is greater than 4 and X is smaller than 10 then if it executes the code block.

```python
print('begin program')
x = 6
if x > 4:
    if x < 10:
         print('Begin codeblock')
         print('x is in range')
         print('end codeblock')
         print('continue program')
```

Now they are on two lines, but they can be on a single line:

```python
if x > 4 and x < 10:
    print('your code')
```

If else

If can be combined with an else clause. That will run the code, if it doesn't meet the conditions. So if x is not a smaller or equal to one or greater than 10 execute the other block.

```python
x = 15
if x >= 1 and x < 10:
    print('In range')
else:
    print('Other code block')
```

So if we have a more simple if statement you can also use the else.

```python
x = 5
if x < 5:
    print('x smaller than 5')
else:
    print('x equal or greater than 5')
```

If variables

You can combine multiple variables in your statements. Let me give you an example. If x is greater than zero and Y is greater than zero then run that's called block.

```python
x = 4
y = 6
if x > 0 and y > 0:
    print('run block')
```

Right so then both x and y need to be greater than zero. If you run it and one of them doesn't meet that condition(s) of the variable(s).

Exercises

1. Why use if statements?
2. Which symbol is used at the end of an if statement?
3. Create a program that shows the text 'lower than 10' when a variable x is lower than ten
4. How to combine if statements?
5. Which operators can be used in an if statement?
6. Create a program that asks for age, and shows 'error' when it's lower than 0 or higher than 120
7. What is the purpose of else?
8. What is the purpose of elif?
9. How many lines can be in an if statement?
10. Can if statements be inside an if statement?
11. Can a for loop be inside an if statement?
12. Can an if statement be inside a for loop?
13. What is the difference between selection and iteration?
14. Are if statements unique to the Python language?
15. Create a program that asks for a password and uses an if statement to verify
16. Create a program that asks the user for a number, and keeps asking until it's correct
17. How many times can you repeat elif?
18. Can you repeat the same if statement?
19. Some programming languages have a switch statement, is there such a thing in Python?
20. Can if statements be used to repeat code?
21. Can you use an if statement in the Python shell?
22. Can if statements be used to check the operating system?
23. Can if statements be used to validate inputs?
24. Why does a line inside an if statement needs four spaces before it?
25. Why does Python use spaces over tabs?

Chapter 8

Files

This chapter will be all about files. You'll be able to write files, read files, append to files. By the end of this chapter you can make programs that use files.

Write file

Writing a file can be done in several steps.

- the first step so actually open the file so specify the file name and at once who writes the file.
- write the file
- finally close the file.

```
fobj = open("textfile.txt", "w+")
fobj.write("Hello World")
fobj.close()
```

So if you run the file doesn't show anything, but look at the files in the directory.

Do you want to write multiple lines?

If you do so, everything is on the same line. So what you want to do is to add the new line character. Then everything will appear on each line.

```
fobj = open("textfile.txt", "w+")
fobj.write("Hello World\n")
fobj.write("Hello World\n")
fobj.write("Hello World\n")
fobj.close()
```

Append to file

If you run this program you see it overwrites. Each time the file is recreated. Now what if you have an existing file, you want to add to the file.
Instead you can do a small change:

```python
fobj = open("textfile.txt", "a")
fobj.write("Hello World\n")
fobj.write("Hello World\n")
fobj.write("Hello World\n")
fobj.close()
```

Do you see what changed?

To append to a file, all you have to do is change the parameter when opening the file. Run it again and you'll see the text is added to the end of the file.

Read file

In Python there are two ways to open files. One create file object and call open function. With "r" for read as read parameter. And say file object readlines.

```python
fobj = open("textfile.txt","r")
lines = fobj.readlines()
print(lines)
```

Use the "r" parameter in there for read, you don't want to put "w" or "a" in there. You also want to close the file when you finished using it.

```python
fobj.close()
```

So this is one way to open files in Python. Another way to open files is using the with keyword So like this:

```python
with open("testfile.txt") as f:
    lines = f.readlines()
    print(lines)
```

This will close the file automatically. Both ways are fine, they are just two different ways to get the same data. So it's whichever you prefer. For the user of the program it doesn't matter which way is used.

Write variables to file

Let's write some actual data to a file. So we start with lists and say a list of some numbers and then we want to write all of these to a file.

```python
x = [2,3,4,5,6,7]
fobj = open("testfile.txt","w")
for element in x:
    fobj.write(str(element) + "\n")
fobj.close()
```

Run it, open the file all of the data has been written. The whole list has been written to the file so you can remove the comma. This works with any list. You can also write a single variable to a file

```python
x = 3
fobj = open("testfile.txt","w")
fobj.write(str(x) + "\n")
fobj.close()
```

Exercises

1. What does open() do?
2. What is the meaning of “w+” ?
3. What is the meaning of “a” ?
4. What is the meaning of “w” ?
5. Does Python automatically close files?
6. Why use the keyword *with*?
7. Does *write()* add a newline?
8. How can you add a newline?
9. Write a program that creates a new file with 3 lines
10. Can you write numbers to a file, if so how?
11. Can you write a list, if so how?
12. Create a program that asks for a filename, reads it and shows the file contents on the screen
13. Create a program that compares two files for equality
14. How can you check if a file exists?
15. Is it necessary to check if a file exists before reading?
16. Can you write tabs to a file?
17. What is the difference between “r” and “r+b”?
18. How to delete a file?
19. How to list a files in a directory?
20. How do you append to an existing file?
21. How to get the current directory?
22. Get the file creation time and modification time
23. How to get the file size in Python?
24. How to move a file?
25. List a files in a directory with the .txt extension

Answers

Chapter 1

1. Yes, you can run Python online. You can do so by visiting a website and typing your Python code, then click run.

Most of the times this doesn't support installing modules.

Websites to run Python online

- https://www.python.org/shell/
- https://repl.it/languages/Python3
- https://pynative.com/online-python-code-editor-to-execute-python-code/
- https://onecompiler.com/python

2. Yes, you can. This is the default for Python programming. There are two ways to go about this:

- Python on your computer. For Linux and Mac, Python is already installed. Open a terminal and type *python* or *python3*. Then press Ctrl+D to quit.
- a Python IDE like PyCharm

3. At the time of writing it's 3.9.0. But you can check the latest version by going to https://www.python.org/downloads/
4. There are several methods to run Python.

Using terminal

One way is by opening a terminal and simply starting the python interpreter. You can do so by typing *python* or *python3*.

You then see the symbols >>>. You can type Python instructions here.

```
$ python3
Python 3.8.2 (default, Mar 13 2020, 10:14:16)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World")
Hello World
>>>
```

You can also create a file with the extension *.py*. Then type the command

```
python program.py
```

Using IDE

You can use an IDE to develop your Python programs. Some programs include

- https://code.visualstudio.com/
- https://www.jetbrains.com/pycharm/

5. You can use any text editor, even notepad. But commonly programmers use a development IDE.
6. The file extension *.py*
7. Create a file hello.py then the contents

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

Type the command:

```
python hello.py
```

On Ubuntu Linux,

```
python3 hello.py
```

8. New file name.py with the data:

```
print("John")
print("Doe")
```

Then the command:

```
python3 name.py
```

You can do this too:

```python
print("John \nDoe")
```

9. There is no difference, you can use both
10. You can use this program:

```python
print("   **")
print("  ****")
print(" ******")
print("********")
print("   **")
print("   **")
```

11. You can draw a rectangle using this code:

```python
print("**********")
print("*        *")
print("*        *")
print("*        *")
print("*        *")
print("**********")
```

12. You can draw a houes using symbols:

```python
print("    **")
print("   *  * ")
print("  *    *")
print(" *      *")
print("*        *")
print("**********")
print("*        *")
print("*        *")
print("*        *")
print("*        *")
print("**********")
```

13. You can do something like this:

```
print("         *  **********  **********")
print("         *  *        *  *         ")
print("         *  *        *  *         ")
print("         *  *        *  *         ")
print("*        *  *        *  *         ")
print("*        *  *        *  **********")
print("*        *  *        *  *         ")
print("*        *  *        *  *         ")
print("*        *  *        *  *         ")
print("*        *  *        *  *         ")
print("**********  **********  **********")
```

14. You can use f-strings like this:

```
>>> age = 10
>>> print(f"My age is {age} years old")
My age is 10 years old
```

You can use this method too:

```
>>> print("My age is", age, "years old")
My age is 10 years old
```

15. You can output it with an f-string:

```
>>> name = "Tiffany"
>>> print(f"Hello {name}")
Hello Tiffany
```

16. One way is:

```
print("$ € £")
```

You can use unicode:

```
>>> print("\u20ac \u0024 \u00A3")
€ $ £
```

Another way to use unicode is by using the name:

```
>>> print "Please pay %s"%(u"\N{euro sign}")
Please pay €
>>> print "Please pay %s"%(u"\N{dollar sign}")
Please pay $
>>> print "Please pay %s"%(u"\N{pound sign}")
Please pay £
```

17. You can output epsilon like this:

```
>>> print("ε")
```

18. False, Python has been around for decades. It first appeared in 1990, which is 30 years ago. But Python comes in many versions, you can check your version with the command

```
python --version
```

19. Python needs to be installed. But you can convert a Python program to a native executable.

You need a tool for that, there are several tools that can do this:

- cx_Freeze for Windows, Linux, and Mac OS X (Python 2.7, 3.x)
- pyinstaller for Windows, Linux, and Mac OS X (Python 2.7, 3.4-3.7)
- bbfreeze for Windows and Linux (Python 2.4-2.7)
- py2exe for Windows (Python 2.6, 2.7)
- py2exe for Windows (Python 3.3-3.5)
- pytoexe for Windows (Python 2, 3)
- Freeze for Linux and maybe Mac OS X (Python 2.x)
- py2app for Mac OS X (Python 2.x)

20. The file extension .py
21. There are many:

- Python programs have an easy syntax
- Quick to create
- Can use for all kinds of domains (web, robotics etc)
- Large community
- No low-level management (memory etc)

22. Python does not need a virtual machine and has an easier syntax
23. You can do it this way:

```
x = 52
y = 48
print(x+y)
```

24. A possible solution:

```
x = 30
y = 10
z = x-y
print(z)
```

25. An IDE gives you many advantages:

- syntax highlighting
- open multiple files at once
- debugging features
- breakpoints
- file explorer

Chapter 2

1. No, the variables x and X are different variables.
2. The output of the program is:

```
3
4
```

3. The data type. `x = 3` is an integer, `x = 3.0` is a float.

```
>>> x = 3
>>> type(x)
<class 'int'>
>>>
>>> x = 3.0
>>> type(x)
<class 'float'>
>>>
```

5. No, variables cannot start with $

```
SyntaxError: invalid syntax
```

6. You can, but you shouldn’t because it’s uncommon
7. No, variable names cannot start with the % character
8. No, in Python ~ is a negative operator
9. It converts text variable to a string variable

```
>>> int("5")
5
```

10. Yes, this is a valid variable name
11. Yes, this is valid
12. This is valid too
13. You can do it like this

```
>>> vat = 1.21
>>> price = 13
>>> total_price = price*vat
>>> total_price
15.73
>>>
```

where 1.21 is 21% VAT

14. A bmi calculator below

```
>>> weight = 60
>>> height = 1.78
>>> bmi = weight / (height*height)
>>> bmi
18.93700290367378
>>>
```

15. A program to do that is:

```
>>> km = 80
>>> km_to_miles = 0.6213712
>>> miles = km * km_to_miles
>>> miles
49.709696
```

16. For miles to km:

```
>>> miles = 49.70
>>> miles_to_km = 1.609344
>>> km = miles * miles_to_km
>>> km
79.98439680000001
```

17. To calculate square

```
>>> x = 4
>>> x*x
16
>>> square = x*x
>>> square
16
```

18. Python throws an exception and ends the program

```
>>> 2/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>>
```

19. Yes, Python supports negative numbers
20. If you use a modern version of Python it is:

```
>>> 7/2
3.5
```

If it’s different, check your Python version

21. You can like this:

```
>>> x,y,z = 3,4,5
>>> x
3
>>> y
4
>>> z
5
>>>
```

22. You can swap variables like this:

```
>>> x,y,z = 3,4,5
>>> x
3
>>> y
4
>>> z
5
>>>
```

23. Because the data type is different. Python guesses the data type. If a number has a dot, it means it could be different than zero like 3.1, 3.2, 3.3 and so on.
24. Because Python programs can be very large. They can be executed multiple times and shared online
25. Yes, you can change the value

Chapter 3

1. Textual data in Python is handled with str objects, or strings. Strings are immutable sequences of Unicode code points. String literals are written in a variety of ways:

- Single quotes: ‘allows embedded “double” quotes’
- Double quotes: “allows embedded ‘single’ quotes”.
- Triple quoted: ‘”Three single quotes’’’, “””Three double quotes”””

Triple quoted strings may span multiple lines - all associated whitespace will be included in the string literal.

2. The minimum length is zero
3. The program demonstrates it:

```python
name = input("What is your name: ")
print("Your name is " + name)
```

4. The program can be like this:

```python
name = input("What is your name: ")
addr = input("What is your address: ")
city = input("What is your city: ")
phone = input("What is your number: ")

print(f"Name {name} address {addr} city {city} phone {phone}")
```

5. a possible solution is:

```python
s = 'a' + 'p' + 'p' + 'l' + 'e'
print(s)
```

6. Its a newline character
7. It inserts a tab space
8. The function *raw_input()* is deprecated. You should use *input()* for the latest versions of Python.
9. Because it's a function. A function is called by adding the brackets
10. You can use the replace function

```python
s = "Hello World"
s = s.replace("World","Python")
print(s)
```

11. No, because brackets call the function.
12. You can use the + symbol

```python
s = "good "
print(s + "morning")
print(s + "afternoon")
print(s + "evening")
```

13. Because the quotes ' are not accepted
14. "helloworld"
15. The output is:

```
SyntaxError: invalid syntax
```

You can't type 1x

16. hellohellohello
17. aaaaabbb
18. hollo
19. gollo

21. You can output it like this

```
print("What's up?")
print('What\'s up?')
```

22. the price is $5.50
23. hello\n
24. The output is:

```
a        b        c
5        6        7
```

25. The output is:

NameError: name 'x' is not defined

Chapter 4

1. A list is a collection.
2. No, a list can contain any type of object including strings, tuples, lists, integers, floats, characters
3. It's a tuple, not a list
4. A list of numbers

```
n [ 1,2,3,4 ]
```

5. To show the maximum number:

```python
print(max(n))
```

6. To show the minimum number

```python
print(min(n))
```

7. The len() function can be used

```python
n = [1,2,3,4,5]
print(len(n))
```

8. List of friends

```python
names = [ "Alice","Wendy", "Gina" ]
```

9. To show the first item

```python
items = [ "Cola", "Fanta", "Milk" ]
print(items[0])
```

10. The length of the list

```python
print(len(items))
```

11. The last item

```python
print(items[-1])
```

12. List of fruits

```python
fruits = [ "Strawberry", "Banana", "Melon" ]
fruits.append("apple")
```

13. To add items

```python
fruits.append("melon")
fruits.append("lemon")
```

14. Yes, you can add two lists together
15. To delete the first item

```
del items[0]
```

16. The brackets are wrong `x[0]`
17. A comma is placed without value in between
18. Yes, this program is correct
19. Yes this is correct
20. No, x is a list and y is a value
21. No, you can't delete from an empty list. This shows the following error:

```
IndexError: list assignment index out of range
```

22. The output is

```
3
```

23. It's a list of lists. It has two lists within the list, you can read it like this

```
x = [ [1,2,3],
      [1,2,3] ]
```

24. The output is 1
25. The output is:

```
5.0
1.0
4.5
```

Chapter 5

1. A tuple

```
items = (1,2,3)
```

2. A list is mutable, a tuple is not. Once defined you can't change a tuple contents
3. You can do it like this:

```
>>> t = (1,2,3)
>>> l = list(t)
>>> l
[1, 2, 3]
```

4. To convert list to a tuple

```
>>> m = tuple(l)
>>> m
(1, 2, 3)
>>>
```

5. You can print using a for loop

```
>>> items = [1,2,3,4,5]
>>> for item in items:
...     print(item)
...
1
2
3
4
5
>>>
>>> t = ('a','b','c')
>>> for item in t:
...     print(item)
...
a
b
c
>>>
```

6. A dictionary is a one to one mapping, of key to value.
7. A key is used to get values
8. Yes, for newer versions of Python a dictionary is ordered
9. Yes, for newer versions
10. Yes, for newer versions
11. Output values in a dictionary

```python
print(dictionary.values()) #prints values
```

12. Output keys in a dictionary

```python
print(dictionary.keys()) #prints keys
```

13. Code below

```python
li = [1,2,3]
del li[0]
```

14. Code below

```python
li = (1,2,3)
del li[0]
```

15. Code below

```python
mydic = {}
mydic['key1'] = 'value1'
mydic['key2'] = 'value2'
mydic['key3'] = 'value3'

del mydic['key1']
```

16. Yes, a list is mutable
17. No, a tuple is not mutable
18. Yes, a dictionary is mutable
19. To output as a table

```python
dic={'Tim':30, 'Kate':25,'Wendy':20}
print('Name\tAge')
for name, age in dic.items():
    print('{}\t{}'.format(name, age))
```

20. Yes it can
21. The definitions are shown below

```python
list1 = [1,2,3]
tuple1 = (1,2,3)
dict1 = {1:'one',2:'two',3:'three'}
```

22. Yes, a list can contain any type of data type
23. It can be like this:

```python
names = []

name = input("Enter name: ")
names.append(name)

name = input("Enter name: ")
names.append(name)

name = input("Enter name: ")
names.append(name)
```

24. You can do it like this:

```python
dict = {}

key = input("key: ")
value = input("value: ")

dict[key] = value
print(dict)
```

25. You can output key,values like this:

```python
mydict = {}
mydict['alice'] = 33
mydict['bob'] = 44
mydict['frank'] = 20

for key, value in mydict.items() :
    print(key, value)
```

Chapter 6

1. To repeat one or more statements
2. After the indention ends. Indention is always four spaces

```
for loop
....code in block
....code in block

# for loop ended
```

3. No, you can use it for input too
4. Yes, you can do that this way:

```
for ...
    for ...
        .....
```

5. Grouping statements by using spaces. In Python each group must have exactly four spaces.
6. Because otherwise you wouldn't repeat any group of statements
7. Yes, you can

```
for item in mylist:
    print(item)
```

8. Yes, in the same way as a list
9. Yes, but this works a bit different because a dictionary has key, value pairs.

To iterate over keys:

```
a_dict = {'color': 'red', 'fruit': 'berry', 'pet': 'cat'}
for key in a_dict:
    print(key)
```

To output the key and value:

```
a_dict = {'color': 'red', 'fruit': 'berry', 'pet': 'cat'}
for key in a_dict:
    print(key, '->', a_dict[key])
```

You can also use the function *items()*:

```python
for key, value in a_dict.items():
    print(key, '->', value)
```

10. Yes, you can. An example is shown below:

```python
mylist = [[1,2,3],[7,5,4]]
for i in range(0,len(mylist)):
    for j in range(0,len(mylist[0])):
        print(mylist[i][j])
```

You can do it like this too:

```python
>>> for sublist in mylist:
...     for j in range(0,len(mylist[0])):
...         print(sublist[j])
```

11. The range() function is used to generate a sequence of numbers
12. A while loop repeats code. A “While” Loop is used to repeat a specific block of code an unknown number of times, until a condition is met.
13. You can do that like this:

```python
mylist = [1,2,3,4,5,6,7]
for item in mylist:
    print(item)
```

If you want to use an index

```python
for i in range(0,len(mylist)):
    print(mylist[i])
```

14. If you use a while loop, you can do this:

```python
a = [1,2,3]
while a:
    print(a.pop(-1))
```

To change order, you can do this:

```
a = [1,2,3]
while a:
    print(a[0])
    del a[0]
```

In both cases the list is empty after completing the loop. You should use a for loop when iterating over a list

15. This means *not* and can be used as comparison operator in a condition
16. Yes, you can:

```
while x < 10 and y < 10:
```

17. Yes

```
s = "hello world"
for c in s:
    print(c)
```

18. For even numbers

```
>>> for i in range(0,10,2):
...     print(i)
```

19. For odd numbers

```
>>> for i in range(1,10,2):
...     print(i)
```

20. You can do it with a while loop

```
>>> cmd = ""
>>> while cmd != "quit":
...     cmd = input("cmd: ")
...
```

21. With the range function you can d othis:

```
>>> for i in range(0,101):
...     print(i)
```

22. A combination of f-strings and for loop solves it:

```
>>> for i in range(0,11):
...     print(f" 5 x {i} = {5*i}")
```

23. The operators greater than >, smaller than <, equal ==, greater or equal >=, smaller or equal <=.
24. Infinite
25. Infinite

Chapter 7

1. To execute code conditionally
2. In Python that's the colon (:)
3. The program

```
x = 5
if x < 5:
    print('lower than 10')
```

4. Using the *and* keyword

```
if this *and* that:
```

5. The operators *and*, *or*, *not*, >=, <=, >, <, ==.
6. The program

```
age = input("age: ")
if age < 0 or age > 120:
    print('error')
```

7. To catch **all** other scenarios

```python
if raining:
    print('take umbrella')
else:
    print('take sunglasses')
```

8. To catch an other scenario

```python
if x > 10:
    ....
elif x > 5:
    ....
elif x > 1:
    ....
```

9. Infinite
10. Infinite
11. Yes, it can
12. Yes, it can
13. selection chooses if to run code, iteration repeats code
14. No, many programming languages have if statements including Java, C++, C, Perl, PHP etc
15. A simple program

```python
pwd = input("Enter password: ")
while pwd != 'okay':
    pwd = input("Enter password: ")
```

16. The program

```python
x = input("Guess the number: ")
while x != 5:
    x = input("Guess the number: ")
```

17. Infinite times
18. Yes, you can
19. No, there is no switch statement in Python
20. No, to repeat code you must use a loop
21. Yes, it can be used in the Python shell too. There is no difference between the Python shell and running from code
22. Yes, you can do that

```
>>> import os
>>> if os.sys.platform == 'linux':
...     print('uses linux')
... else:
...     print('uses something else')
...
uses linux
>>>
```

23. Yes, see answer 15 and 16
24. Because that's how Python sees code blocks. Other program languages use the words *begin*, *end* or the symbols **
25. Because spaces are the same on every operating system

Chapter 8

1. It opens a file and returns the file object
2. Write and overwrite
3. Append, add to the file
4. Write but don't overwrite
5. You should close the files yourself. By calling the *.close()* function on the file object
6. Because then you don't have to type *.close()*
7. It writes to the file, but it doesn't add newlines by default
8. By calling *.write("\n")* or call *.writelines(["Line 1", "Line 2"])*
9. The program below

```
f = open("test.txt","w")
f.write("line 1\n")
f.write("line 2\n")
f.write("line 3\n")
f.close()
```

10. Yes you can. Program:

```
f = open("test.txt","w+")
num = 5
f.write('{}'.format(num))
f.close()
```

11. Yes you can.

```python
myList = ["New York", "London", "Paris", "New Delhi"]
fobj=open('output.txt','w')

for element in myList:
     fobj.write(element)
     fobj.write('\n')
fobj.close()
```

12. The program

```python
# Get filename
name = input("Enter filename: ")

# Open file object
fobj = open(name,"r")

# Read data
data = fobj.readlines()

for line in data:
    print(line, end='')
```

13. The program

```python
file1 = open("test.txt","r")
file2 = open("test.txt","r")

# Read data
data = file1.read()
data2 = file2.read()

if data != data2:
    print("Not equal files")
else:
    print("Equal file content")
```

14. You can use the os module

```python
import os.path

if os.path.isfile('filename.txt'):
    print ("File exist")
else:
    print ("File not exist")
```

15. Yes, otherwise it throws an exception: **FileNotFoundError**:
16. Yes, with the \t character
17. *r* reads the file in text mode, *r+b* reads it in binary (computer) mode
18. You can do it like this

```python
import os
os.remove("file.txt")
```

19. Using the os or glob module

```python
import glob
print(glob.glob("/home/frank/*.txt"))
```

20. By opening with the *a* flag
21. Using the os module

```python
import os
currentDirectory = os.getcwd()
```

22. The os module will help you

```python
import os.path, time
print("last modified: %s" % time.ctime(os.path.getmtime(file)))
print("created: %s" % time.ctime(os.path.getctime(file)))
```

23. Using *os.path.getsize*

```python
import os
b = os.path.getsize("/path/isa_005.mp3")
```

24. os.rename() or shutil.move()
25. You can use the glob module for that, see answer 19