

PYTHON TUTORIAL

What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

Good to know

- The most recent major version of Python is Python 3. However, Python 2, although not being updated with anything other than security updates, is still quite popular.
- In this tutorial Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment (IDE), such as Thonny, Pycharm, Netbeans

or Eclipse which are particularly useful when managing larger collections of Python files.

Python Syntax compared to other programming languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

Python Install

Many PCs and Macs will have python already installed.

To check if you have python installed on a Windows PC, search in the start bar for Python or run the following on the Command Line (cmd.exe):

```
C:\Users\Your Name>python --version
```

To check if you have python installed on a Linux or Mac, then on linux open the command line or on Mac open the Terminal and type:

```
python --version
```

If you find that you do not have Python installed on your computer, then you can download it for free from the following website: <https://www.python.org/>

Python Quickstart

Python is an interpreted programming language, this means that as a developer you write Python (.py) files in a text editor and then put those files into the python interpreter to be executed.

The way to run a python file is like this on the command line:

```
C:\Users\Your Name>python helloworld.py
```

Where "helloworld.py" is the name of your python file.

Let's write our first Python file, called helloworld.py, which can be done in any text editor.

```
helloworld.py  
print("Hello, World!")
```

Simple as that. Save your file. Open your command line, navigate to the directory where you saved your file, and run:

```
C:\Users\Your Name>python helloworld.py
```

The output should read:

```
Hello, World!
```

The Python Command Line

To test a short amount of code in python sometimes it is quickest and easiest not to write the code in a file. This is made possible because Python can be run as a command line itself.

Type the following on the Windows, Mac or Linux command line:

```
C:\Users\Your Name>python
```

Or, if the "python" command did not work, you can try "py":

```
C:\Users\Your Name>py
```

From there you can write any python, including our hello world example from earlier in the tutorial:

```
C:\Users\Your Name>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, World!")
```

Which will write "Hello, World!" in the command line:

```
C:\Users\Your Name>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, World!")
Hello, World!
```

Whenever you are done in the python command line, you can simply type the following to quit the python command line interface:

```
exit()
```

Execute Python Syntax

As we learned in the previous page, Python syntax can be executed by writing directly in the Command Line:

```
>>> print("Hello, World!")  
Hello, World!
```

Or by creating a python file on the server, using the .py file extension, and running it in the Command Line:

```
C:\Users\Your Name>python myfile.py
```

Python Indentation

Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.

Example

```
if 5 > 2:  
    print("Five is greater than two!")
```

Python will give you an error if you skip the indentation:

Example

Syntax Error:

```
if 5 > 2:  
print("Five is greater than two!")
```

Example

```
if 5 > 2:  
    print("Five is greater than two!")  
if 5 > 2:  
    print("Five is greater than two!")
```

You have to use the same number of spaces in the same block of code, otherwise Python will give you an error:

Syntax Error:

```
if 5 > 2:  
    print("Five is greater than two!")  
    print("Five is greater than two!")
```

Python Data Types

Built-in Data Types

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

Text Type:	<code>str</code>
Numeric Types:	<code>int</code> , <code>float</code> , <code>complex</code>
Sequence Types:	<code>list</code> , <code>tuple</code> , <code>range</code>
Mapping Type:	<code>dict</code>
Set Types:	<code>set</code> , <code>frozenset</code>
Boolean Type:	<code>bool</code>
Binary Types:	<code>bytes</code> , <code>bytearray</code> , <code>memoryview</code>
None Type:	<code>NoneType</code>

Getting the Data Type

You can get the data type of any object by using the `type()` function:

Example

Print the data type of the variable x:

```
x = 5  
print(type(x))
```

```
x = 5  
print(type(x))
```

```
<class 'int'>
```

Setting the Data Type

In Python, the data type is set when you assign a value to a variable:

Example	Data Type
<code>x = "Hello World"</code>	str
<code>x = 20</code>	int
<code>x = 20.5</code>	float
<code>x = 1j</code>	complex
<code>x = ["apple", "banana", "cherry"]</code>	list
<code>x = ("apple", "banana", "cherry")</code>	tuple
<code>x = range(6)</code>	range
<code>x = {"name" : "John", "age" : 36}</code>	dict
<code>x = {"apple", "banana", "cherry"}</code>	set
<code>x = frozenset({"apple", "banana", "cherry"})</code>	frozenset
<code>x = True</code>	bool
<code>x = b"Hello"</code>	bytes
<code>x = bytearray(5)</code>	bytearray
<code>x = memoryview(bytes(5))</code>	memoryview
<code>x = None</code>	NoneType

```
x = "Hello World"
```

```
#display x:  
print(x)
```

```
#display the data type of x:  
print(type(x))
```

```
Hello World  
<class 'str'>
```


Python Numbers

There are three numeric types in Python:

- `int`
- `float`
- `complex`

Variables of numeric types are created when you assign a value to them:

Example

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

To verify the type of any object in Python, use the `type()` function:

Example

```
print(type(x))
print(type(y))
print(type(z))
```

```
x = 1
y = 2.8
z = 1j

print(type(x))
print(type(y))
print(type(z))
```

```
<class 'int'>
<class 'float'>
<class 'complex'>
```

Int

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

Example

Integers:

```
x = 1
y = 35656222554887711
z = -3255522
```

```
print(type(x))
print(type(y))
print(type(z))
```

```
x = 1
y = 35656222554887711
z = -3255522
```

```
print(type(x))
print(type(y))
print(type(z))
```

```
<class 'int'>
<class 'int'>
<class 'int'>
```

Float

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

Example

Floats:

```
x = 1.10
y = 1.0
z = -35.59
```

```
print(type(x))
print(type(y))
print(type(z))
```

```
x = 1.10
y = 1.0
z = -35.59

print(type(x))
print(type(y))
print(type(z))
```

```
<class 'float'>
<class 'float'>
<class 'float'>
```

Float can also be scientific numbers with an "e" to indicate the power of 10.

Example

Floats:

```
x = 35e3
y = 12E4
z = -87.7e100

print(type(x))
print(type(y))
print(type(z))
```

```
x = 35e3
y = 12E4
z = -87.7e100

print(type(x))
print(type(y))
print(type(z))
```

```
<class 'float'>
<class 'float'>
<class 'float'>
```

Complex

Complex numbers are written with a "j" as the imaginary part:

Example

Complex:

```
x = 3+5j
y = 5j
z = -5j

print(type(x))
print(type(y))
print(type(z))
```

```
x = 3+5j
y = 5j
z = -5j

print(type(x))
print(type(y))
print(type(z))
```

```
<class 'complex'>
<class 'complex'>
<class 'complex'>
```

Type Conversion

You can convert from one type to another with the `int()`, `float()`, and `complex()` methods:

Example

Convert from one type to another:

```
x = 1 # int
y = 2.8 # float
z = 1j # complex

#convert from int to float:
a = float(x)

#convert from float to int:
b = int(y)

#convert from int to complex:
c = complex(x)

print(a)
print(b)
print(c)

print(type(a))
print(type(b))
print(type(c))
```

```
#convert from int to float:
x = float(1)

#convert from float to int:
y = int(2.8)

#convert from int to complex:
z = complex(1)

print(x)
print(y)
print(z)

print(type(x))
print(type(y))
print(type(z))
```

```
1.0
2
(1+0j)
<class 'float'>
<class 'int'>
<class 'complex'>
```

Note: You cannot convert complex numbers into another number type.

Random Number

Python does not have a `random()` function to make a random number, but Python has a built-in module called `random` that can be used to make random numbers:

Example

Import the `random` module, and display a random number between 1 and 9:

```
import random

print(random.randrange(1, 10))
```

```
import random

print(random.randrange(1, 10))
```

3

Python Casting

Specify a Variable Type

There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Casting in python is therefore done using constructor functions:

- `int()` - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)
- `float()` - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
- `str()` - constructs a string from a wide variety of data types, including strings, integer literals and float literals

Example

Integers:

```
x = int(1)    # x will be 1
y = int(2.8)  # y will be 2
z = int("3")  # z will be 3
```

```
x = int(1)
y = int(2.8)
z = int("3")
print(x)
print(y)
print(z)
```

```
1
2
3
```

Example

Floats:

```
x = float(1)      # x will be 1.0
y = float(2.8)    # y will be 2.8
z = float("3")    # z will be 3.0
w = float("4.2") # w will be 4.2
```

```
x = float(1)
y = float(2.8)
z = float("3")
w = float("4.2")
print(x)
print(y)
print(z)
print(w)
```

```
1.0
2.8
3.0
4.2
```

Example

Strings:

```
x = str("s1") # x will be 's1'
y = str(2)    # y will be '2'
z = str(3.0)  # z will be '3.0'
```

```
x = str("s1")
y = str(2)
z = str(3.0)
print(x)
print(y)
print(z)
```

```
s1
2
3.0
```

Python If ... Else

Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

- Equals: `a == b`
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`

These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the `if` keyword.

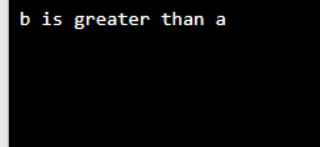
Example

If statement:

```
a = 33
b = 200
if b > a:
    print("b is greater than a")
```

```
a = 33
b = 200

if b > a:
    print("b is greater than a")
```



b is greater than a

In this example we use two variables, `a` and `b`, which are used as part of the if statement to test whether `b` is greater than `a`. As `a` is 33, and `b` is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

Indentation

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

Example

If statement, without indentation (will raise an error):

```
a = 33
b = 200
if b > a:
print("b is greater than a") # you will get an error
```

```
a = 33
b = 200

if b > a:
print("b is greater than a")
```

```
File "demo_if_error.py", line 4
    print("b is greater than a")
    ^
IndentationError: expected an indented block
```

Elif

The `elif` keyword is python's way of saying "if the previous conditions were not true, then try this condition".

Example

```
a = 33
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
```

```
a = 33
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
```

```
a and b are equal
```

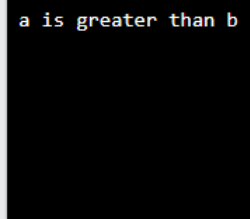
Else

The `else` keyword catches anything which isn't caught by the preceding conditions.

Example

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

A terminal window with a black background and white text. The text reads "a is greater than b".

a is greater than b

Short Hand If

If you have only one statement to execute, you can put it on the same line as the if statement.

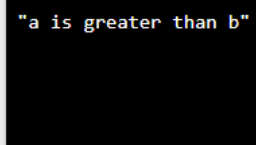
Example

One line if statement:

```
if a > b: print("a is greater than b")
```

```
a = 200
b = 33

if a > b: print("a is greater than b")
```

A terminal window with a black background and white text. The text reads "a is greater than b".

"a is greater than b"

Short Hand If ... Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

Example

One line if else statement:

```
a = 2
b = 330
print("A") if a > b else print("B")
```

```
a = 2
b = 330

print("A") if a > b else print("B")
```



This technique is known as **Ternary Operators**, or **Conditional Expressions**.

You can also have multiple else statements on the same line:

Example

One line if else statement, with 3 conditions:

```
a = 330
b = 330
print("A") if a > b else print("=") if a == b else print("B")
```

```
a = 330
b = 330

print("A") if a > b else print("=") if a == b else print("B")
```



And

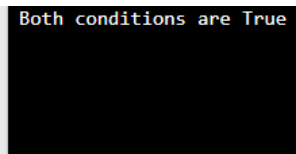
The `and` keyword is a logical operator, and is used to combine conditional statements:

Example

Test if `a` is greater than `b`, AND if `c` is greater than `a`:

```
a = 200
b = 33
c = 500
if a > b and c > a:
    print("Both conditions are True")
```

```
a = 200
b = 33
c = 500
if a > b and c > a:
    print("Both conditions are True")
```

A terminal window with a black background and white text. The text reads "Both conditions are True".

Or

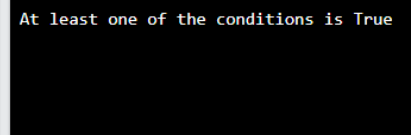
The `or` keyword is a logical operator, and is used to combine conditional statements:

Example

Test if `a` is greater than `b`, OR if `a` is greater than `c`:

```
a = 200
b = 33
c = 500
if a > b or a > c:
    print("At least one of the conditions is True")
```

```
a = 200
b = 33
c = 500
if a > b or a > c:
    print("At least one of the conditions is True")
```

A terminal window with a black background and white text. The text reads "At least one of the conditions is True".

Nested If

You can have `if` statements inside `if` statements, this is called *nested if* statements.

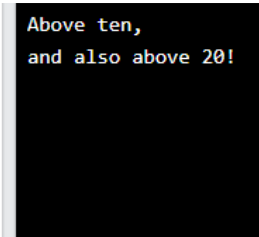
Example

```
x = 41

if x > 10:
    print("Above ten,")
    if x > 20:
        print("and also above 20!")
    else:
        print("but not above 20.")
```

```
x = 41

if x > 10:
    print("Above ten,")
    if x > 20:
        print("and also above 20!")
    else:
        print("but not above 20.")
```



```
Above ten,
and also above 20!
```

The pass Statement

`if` statements cannot be empty, but if you for some reason have an `if` statement with no content, put in the `pass` statement to avoid getting an error.

Example

```
a = 33
b = 200
```

```
if b > a:
    pass
```

```
a = 33
b = 200

if b > a:
    pass

# having an empty if statement like this, would raise an error without the pass
statement
```

BOOL FUCTION

Evaluate Values and Variables

The `bool()` function allows you to evaluate any value, and give you `True` or `False` in return,

Example

Evaluate a string and a number:

```
print(bool("Hello"))  
print(bool(15))
```

```
print(bool("Hello"))  
print(bool(15))
```



```
True  
True
```

Example

Evaluate two variables:

```
x = "Hello"  
y = 15  
  
print(bool(x))  
print(bool(y))
```

```
x = "Hello"  
y = 15  
  
print(bool(x))  
print(bool(y))
```



```
True  
True
```

Most Values are True

Almost any value is evaluated to `True` if it has some sort of content.

Any string is **True**, except empty strings.

Any number is **True**, except **0**.

Any list, tuple, set, and dictionary are **True**, except empty ones.

Some Values are False

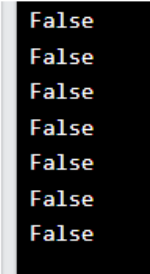
In fact, there are not many values that evaluate to **False**, except empty values, such as **()**, **[]**, **{}**, **""**, the number **0**, and the value **None**. And of course the value **False** evaluates to **False**.

Example

The following will return **False**:

```
bool(False)
bool(None)
bool(0)
bool("")
bool(())
bool([])
bool({})
```

```
print(bool(False))
print(bool(None))
print(bool(0))
print(bool(""))
print(bool(()))
print(bool([]))
print(bool({})
```



```
False
False
False
False
False
False
False
```

One more value, or object in this case, evaluates to **False**, and that is if you have an object that is made from a class with a `__len__` function that returns **0** or **False**:

Example

```
class myclass():
    def __len__(self):
        return 0

myobj = myclass()
print(bool(myobj))
```

```
class myclass():
    def __len__(self):
        return 0

myobj = myclass()
print(bool(myobj))
```

False

Functions can Return a Boolean

You can create functions that returns a Boolean Value:

Example

Print the answer of a function:

```
def myFunction() :
    return True

print(myFunction())

def myFunction() :
    return True

print(myFunction())
```

True

Operator	Name	Example
+	Addition	$x + y$
-	Subtraction	$x - y$
*	Multiplication	$x * y$
/	Division	x / y
%	Modulus	$x \% y$
**	Exponentiation	$x ** y$
//	Floor division	$x // y$

Operator	Name	Example
==	Equal	$x == y$
!=	Not equal	$x != y$
>	Greater than	$x > y$
<	Less than	$x < y$
>=	Greater than or equal to	$x >= y$
<=	Less than or equal to	$x <= y$

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
=	x = 3	x = x 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Python While Loops

Python Loops

Python has two primitive loop commands:

- `while` loops
- `for` loops

- ## The while Loop

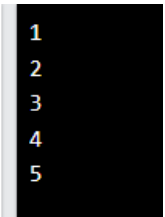
- With the `while` loop we can execute a set of statements as long as a condition is true.

- ### Example

- Print `i` as long as `i` is less than 6:

- ```
i = 1
while i < 6:
 print(i)
 i += 1
```

```
i = 1
while i < 6:
 print(i)
 i += 1
```



```
1
2
3
4
5
```

**Note:** remember to increment `i`, or else the loop will continue forever.

The `while` loop requires relevant variables to be ready, in this example we need to define an indexing variable, `i`, which we set to 1.

# The break Statement

With the `break` statement we can stop the loop even if the while condition is true:

## Example

Exit the loop when i is 3:

```
i = 1
while i < 6:
 print(i)
 if i == 3:
 break
 i += 1
```

```
i = 1
while i < 6:
 print(i)
 if (i == 3):
 break
 i += 1
```



1  
2  
3

# The continue Statement

With the `continue` statement we can stop the current iteration, and continue with the next:

## Example

Continue to the next iteration if i is 3:

```
i = 0
while i < 6:
 i += 1
 if i == 3:
 continue
 print(i)
```

```
i = 0
while i < 6:
 i += 1
 if i == 3:
 continue
 print(i)

Note that number 3 is missing in the result
```

```
1
2
4
5
6
```

## The else Statement

With the `else` statement we can run a block of code once when the condition no longer is true:

### Example

Print a message once the condition is false:

```
i = 1
while i < 6:
 print(i)
 i += 1
else:
 print("i is no longer less than 6")
```

```
i = 1
while i < 6:
 print(i)
 i += 1
else:
 print("i is no longer less than 6")
```

```
1
2
3
4
5
i is no longer less than 6
```

# Python For Loops

## Python Collections (Arrays)

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered, unchangeable\*, and unindexed. No duplicate members.

- [Dictionary](#) is a collection which is ordered\*\* and changeable. No duplicate members.

# Python Lists

```
mylist = ["apple", "banana", "cherry"]
```

## List

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are [Tuple](#), [Set](#), and [Dictionary](#), all with different qualities and usage.

Lists are created using square brackets:

## Example

Create a List:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

```
['apple', 'banana', 'cherry']
```

## List Items

List items are ordered, changeable, and allow duplicate values.

List items are indexed, the first item has index `[0]`, the second item has index `[1]` etc.

# Ordered

When we say that lists are ordered, it means that the items have a defined order, and that order will not change.

If you add new items to a list, the new items will be placed at the end of the list.

**Note:** There are some [list methods](#) that will change the order, but in general: the order of the items will not change.

# Changeable

The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

# Allow Duplicates

Since lists are indexed, lists can have items with the same value:

## Example

Lists allow duplicate values:

```
thislist = ["apple", "banana", "cherry", "apple", "cherry"]
print(thislist)
```

```
thislist = ["apple", "banana", "cherry", "apple", "cherry"]
print(thislist)
```

```
['apple', 'banana', 'cherry', 'apple', 'cherry']
```

# List Length

To determine how many items a list has, use the `len()` function:

## Example



Print the number of items in the list:

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

## List Items - Data Types

List items can be of any data type:

### Example

String, int and boolean data types:

```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
```

```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]

print(list1)
print(list2)
print(list3)
```

```
['apple', 'banana', 'cherry']
[1, 5, 7, 9, 3]
[True, False, False]
```

A list can contain different data types:

### Example

A list with strings, integers and boolean values:

```
list1 = ["abc", 34, True, 40, "male"]
```

```
list1 = ["abc", 34, True, 40, "male"]
print(list1)
```

```
['abc', 34, True, 40, 'male']
```

## type()

From Python's perspective, lists are defined as objects with the data type 'list':

```
<class 'list'>
```

## Example

What is the data type of a list?

```
mylist = ["apple", "banana", "cherry"]
print(type(mylist))
```

```
mylist = ["apple", "banana", "cherry"]
print(type(mylist))
```

```
<class 'list'>
```

## The list() Constructor

It is also possible to use the `list()` constructor when creating a new list.

## Example

Using the `list()` constructor to make a List:

```
thislist = list(("apple", "banana", "cherry")) # note the double round-brackets
print(thislist)
```

```
thislist = list(("apple", "banana", "cherry"))
print(thislist)
```

```
['apple', 'banana', 'cherry']
```