

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 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 a 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.

Python Syntax compared to other programming languages

- Python was designed for readability and had 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.

Example

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

Python Getting Started

Please see the video which is install Python and PyCharm Community Edition.

Grammar = Syntax

2. Python Syntax – Execute Python Syntax, Python Indentation

2.1. Execute Python Syntax:

- Python syntax can be executed by writing directly in the Command Line:

Example:

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

2.2. 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!")
```

The number of spaces is up to you as a programmer, the most common use is four, but it has to be at least one.

Example

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

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

Example Syntax Error:

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

Python Variables

- In Python, variables are created when you assign a value to it:

Example Variables in Python:

```
x = 5
y = "Hello, World!"
print(x)
print(y)
```

Python Comments

Comments can be used to explain Python code.

Comments can be used to make the code more readable.

Comments can be used to prevent execution when testing code.

Creating a Comment

- Comments start with a #, and Python will ignore them:

Example

```
#This is a comment
print("Hello, World!")
```

- Comments can be placed at the end of a line, and Python will ignore the rest of the line:

Example

```
print("Hello, World!") #This is a comment
```

- A comment does not have to be text that explains the code, it can also be used to prevent Python from executing code:

Example

```
#print("Hello, World!")
print("Cheers, Mate!")
```

Multiline Comments

- Python does not really have a syntax for multiline comments.
- To add a multiline comment you could insert a # for each line:

Example

```
#This is a comment  
#written in  
#more than just one line  
print("Hello, World!")
```

Or, not quite as intended, you can use a multiline string.

Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:

Example

```
"""  
This is a comment  
written in  
more than just one line  
"""  
  
print("Hello, World!")
```

3. Python Variables

Variables:

Variables are containers for storing data values.

Creating Variables

Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

Example

```
x = 5
y = "John"
print(x)
print(y)
```

Variables do not need to be declared with any particular *type*, and can even change type after they have been set.

Example

```
x = 4          # x is of type int
x = "Sally"   # x is now of type str
print(x)
```

Casting

If you want to specify the data type of a variable, this can be done with casting.

Example

```
x = str("Hello World") # x will be Hello World
y = int(3)             # y will be 3
z = float(3)           # z will be 3.0

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

You can get the data type of a variable with the `type()` function.

Example

```
x = 5
y = "John"
print(type(x))
print(type(y))
```

Single or Double Quotes?

String variables can be declared either by using single or double quotes:

Example

```
x = "John"
# is the same as
x = 'John'

print(x)
```

Case-Sensitive

Variable names are case-sensitive.

Example

This will create two variables:

```
a = 4
A = "Sally"
#A will not overwrite a

print(a)

print(A)
```

Python - Variable Names

Variable Names

A variable can have a short name (like x and y) or a more descriptive name.
Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- A variable name cannot be any of the Python keywords.

Example

Legal variable names:

```
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

Example

Illegal variable names:

```
2myvar = "John"  
my-var = "John"  
my var = "John"
```

Multi Words Variable Names

Variable names with more than one word can be difficult to read.

There are several techniques you can use to make them more readable:

Camel Case

Each word, except the first, starts with a capital letter:

Example

```
myVariableName = "John1"  
print(myVariableName)
```

Pascal Case

Each word starts with a capital letter:

Example

```
MyVariableName = "John2"  
print(MyVariableName)
```

Snake Case

Each word is separated by an underscore character:

Example

```
my_variable_name = "John3"  
print(my_variable_name)
```

Many Values to Multiple Variables

Python allows you to assign values to multiple variables in one line:

Example

```
x, y, z = "Orange", "Banana", "Cherry"  
print(x)  
print(y)  
print(z)
```

One Value to Multiple Variables

And you can assign the *same* value to multiple variables in one line:

Example

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

Python - Output Variables

Output Variables

The Python `print()` function is often used to output variables.

Example

```
x = "Python is awesome"
print(x)
```

In the `print()` function, you output multiple variables, separated by a comma:

Example

```
x = "Python"
y = "is"
z = "awesome"
print(x, y, z)
```

You can also use the `+` operator to output multiple variables:

Example

```
x = "Python "
y = "is "
z = "awesome"
print(x + y + z)
```

Notice the space character after "Python " and "is ", without them the result would be "Pythonisawesome".

For numbers, the + character works as a mathematical operator:

Example

```
x = 5
y = 10
print(x + y)
```

In the print() function, when you try to combine a string and a number with the + operator, Python will give you an error:

Example

```
x = 5
y = "John"
print(x + y)
```

The best way to output multiple variables in the print() function is to separate them with commas, which even support different data types:

Example

```
x = 5
y = "John"
print(x, y)
```

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

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))
```

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

- غير قيمة X في المثال الاتي طبقاً للجدول اعلاه.

Example:

```
x = "Hello World"

#display x:
print(x)

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

Setting the Specific Data Type

If you want to specify the data type, you can use the following constructor functions:

Example	Data Type
<code>x = str("Hello World")</code>	str
<code>x = int(20)</code>	int
<code>x = float(20.5)</code>	float
<code>x = complex(1j)</code>	complex
<code>x = list(("apple", "banana", "cherry"))</code>	list
<code>x = tuple(("apple", "banana", "cherry"))</code>	tuple
<code>x = range(6)</code>	range
<code>x = dict(name="John", age=36)</code>	dict
<code>x = set(("apple", "banana", "cherry"))</code>	set
<code>x = frozenset(("apple", "banana", "cherry"))</code>	frozenset
<code>x = bool(5)</code>	bool

<code>x = bytes([65, 66, 67, 68, 69])</code>	<code>bytes</code>
<code>x = bytearray(5)</code>	<code>bytearray</code>
<code>x = memoryview(bytes(5))</code>	<code>memoryview</code>

• غير قيمة X في المثال الاتي طبقا للجدول اعلاه.

Example:

```
x = str("Hello World")

#display x:
print(x)

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

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

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

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))
```

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))
```

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

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))
```

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))
```

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))
```

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
print(x,y,z)
```

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
print(x,y,z)
```

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'
print(x,y,z)
```

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

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


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
print(x)
print(y)
print(z)
print(w)
```

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'
print(x)
print(y)
print(z)
```

Python Strings

Strings in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

You can display a string literal with the `print()` function:

Example

```
print("Hello")
print('Hello')
```

Assign String to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

Example

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

Multiline Strings

You can assign a multiline string to a variable by using three quotes:

Example

You can use three double quotes:

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

Or three single quotes:

Example

```
a = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
print(a)
```

Strings are Arrays

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1.

Square brackets can be used to access elements of the string.

Example

Get the character at position 1 (remember that the first character has the position 0):

```
a = "Hello, World!"  
print(a[1])
```

Looping Through a String

Since strings are arrays, we can loop through the characters in a string, with a for loop.

Example

Loop through the letters in the word "banana":

```
for x in "banana":  
    print(x)
```

String Length

To get the length of a string, use the `len()` function.

Example

The len() function returns the length of a string:

```
a = "Hello, World!"  
print(len(a))
```

Check String

To check if a certain phrase or character is present in a string, we can use the keyword `in`.

Example

Check if "free" is present in the following text:

```
txt = "The best things in life are free!"  
print("free" in txt)
```

Use it in an if statement:

Example

Print only if "free" is present:

```
txt = "The best things in life are free!"
if "free" in txt:
    print("Yes, 'free' is present.")
```

Check if NOT

To check if a certain phrase or character is NOT present in a string, we can use the keyword not in.

Example

Check if "expensive" is NOT present in the following text:

```
txt = "The best things in life are free!"
print("expensive" not in txt)
```

Use it in an if statement:

Example

print only if "expensive" is NOT present:

```
txt = "The best things in life are free!"
if "expensive" not in txt:
    print("No, 'expensive' is NOT present.")
```

Python Data Types

Python Strings-

Slicing

You can return a range of characters by using the slice syntax.

Specify the start index and the end index, separated by a colon, to return a part of the string.

Get the characters from position 2 to position 5 (not included):

Example

```
b = "Hello, World!"  
print(b[2:5])
```

Note: The first character has an index of 0.

Slice From the Start

By leaving out the start index, the range will start at the first character:

Get the characters from the start to position 5 (not included):

Example

```
b = "Hello, World!"  
print(b[:5])
```

Slice To the End

By leaving out the *end* index, the range will go to the end:

Get the characters from position 2, and all the way to the end:

Example

```
b = "Hello, World!"  
print(b[2:])
```

Negative Indexing

Use negative indexes to start the slice from the end of the string:

Example

Get the characters:

From: "o" in "World!" (position -5)

To, but not included: "d" in "World!" (position -2):

```
b = "Hello, World!"  
print(b[-5:-2])
```

Python Data Types

Python Strings-

Modify Strings

Python has a set of built-in methods that you can use on strings.

Upper Case

Example

The `upper()` method returns the string in upper case:

```
a = "Hello, World!"  
print(a.upper())
```

Lower Case

Example

The `lower()` method returns the string in lower case:

```
a = "Hello, World!"  
print(a.lower())
```

Remove Whitespace

Whitespace is the space before and/or after the actual text, and very often you want to remove this space.

Example

The `strip()` method removes any whitespace from the beginning or the end:

```
a = " Hello, World! "  
print(a.strip()) # returns "Hello, World!"
```

Replace String

Example

The `replace()` method replaces a string with another string:

```
a = "Hello, World!"  
print(a.replace("H", "J"))
```

Split String

The `split()` method returns a list where the text between the specified separator becomes the list items.

Example

The `split()` method splits the string into substrings if it finds instances of the separator:

```
a = "Hello, World!"  
print(a.split(",")) # returns ['Hello', ' World!']
```

Python Data Types**Python Strings-****String Concatenation**

To concatenate, or combine, two strings you can use the + operator.

Example

Merge variable **a** with variable **b** into variable **c**:

```
a = "Hello"  
b = "World"  
c = a + b  
print(c)
```

Example

To add a space between them, add a " ":

```
a = "Hello"  
b = "World"  
c = a + " " + b  
print(c)
```

Python Data Types

Python Strings-

String Format

As we learned in the Python Variables, we cannot combine strings and numbers like this:

Example

```
age = 36
txt = "My name is John, I am " + age
print(txt)
```

But we can combine strings and numbers by using the `format()` method!

The `format()` method takes the passed arguments, formats them, and places them in the string where the placeholders `{}` are:

Example

Use the `format()` method to insert numbers into strings:

```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))
```

The `format()` method takes unlimited number of arguments, and are placed into the respective placeholders:

Example

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

You can use index numbers `{0}` to be sure the arguments are placed in the correct placeholders:

Example

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```


Python Data Types**Python Strings-****Escape Characters**

To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash \ followed by the character you want to insert.

An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

Example

You will get an error if you use double quotes inside a string that is surrounded by double quotes:

```
txt = "We are the so-called "A" from the north."  
print(txt)
```

To fix this problem, use the escape character \":

Example

```
txt = "We are the so-called \"A\" from the north."  
print(txt)
```

Escape Characters

Other escape characters used in Python:

Code	Result
\'	Single Quote
\\	Backslash
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace
\f	Form Feed
\ooo	Octal value
\xhh	Hex value

Example 1

```
txt = 'It\'s alright.'  
print(txt)
```

output

It's alright.

Example 2

```
txt = "This will insert one \\ (backslash)."  
print(txt)
```

output

This will insert one \ (backslash).

Example 3

```
txt = "Hello\nWorld!"  
print(txt)
```

output

Hello
World!

Example 4

```
txt = "Hello\rWorld!"  
print(txt)
```

output

World!

1. "Hello" is printed.
2. The `\r` causes the cursor to return to the start of the line.
3. " World!" is then printed, starting from the beginning of the line.

So, instead of creating a new line, the carriage return essentially "rewinds" the cursor to the start of the current line. As a result, "World!" is printed over the existing characters, giving the appearance of replacing the space after "Hello."

The output visually looks like:

Example 5

```
txt = "Hello\tWorld!"  
print(txt)
```

output

```
Hello    World!
```

Example 6

```
#This example erases one character (backspace):  
txt = "Hello \bWorld!"  
print(txt)
```

output

```
HelloWorld!
```

Example 7

```
#A backslash followed by three integers will result in a octal value:  
txt = "\110\145\154\154\157"  
print(txt)
```

output

```
Hello
```

Example 8

```
#A backslash followed by an 'x' and a hex number represents a hex value:  
txt = "\x48\x65\x6c\x6c\x6f"  
print(txt)
```

output

```
Hello
```

Python Data Types

Python Booleans

Booleans represent one of two values: `True` or `False`.

Boolean Values

In programming you often need to know if an expression is `True` or `False`.

You can evaluate any expression in Python, and get one of two answers, `True` or `False`.

When you compare two values, the expression is evaluated and Python returns the Boolean answer:

Example

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

When you run a condition in an if statement, Python returns `True` or `False`:

Example

Print a message based on whether the condition is `True` or `False`:

```
a = 200
b = 33

if b > a:
    print("b is greater than a")
else:
    print("b is not greater than a")
```

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))
```

Example

Evaluate two variables:

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

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.

Example

The following will return `True`:

```
bool("abc")
bool(123)
bool(["apple", "cherry", "banana"])
```

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({})
```

Python Operators

Python Operators

Operators are used to perform operations on variables and values.

In the example below, we use the `+` operator to add together two values:

Example

```
print(10 + 5)
```

Python divides the operators into the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Example

```
print("Expressions")
print('Arithmetic Expressions ')
a = 2
b = 3
c = 2.3

d = a + b
print(d) # 5
print(a + b) # 5
print(a + c) # 4.3
print(b / a) # 1.5 # see the future
print(b // a) # 1 # floor division
print(a * c) # 4.6
print(a ** b) # 8 (power)
print(17 % 3) # 2 (modulus)
a += 7 # is the same as a = a + 7
print(a) # 9
# a++ # SyntaxError: invalid syntax
# a-- # SyntaxError: invalid syntax
a += 1
print(a) # 10
a -= 1
print(a) # 9
```

Example

```
print("Python Comparison Operators")
a = 5
b = 2
print (a > b)
```

Example

```
print("Python Comparison Operators")
print('== Is Equal To Example 3 == 5 gives us False')
print('!= Not Equal To Example 3 != 5 gives us True')
print('> Greater Than Example 3 > 5 gives us False')
print('< Less Than Example 3 < 5 gives us True')
print('>= Greater Than or Equal To Example 3 >= 5 give us False')
print('<= Less Than or Equal To Example 3 <= 5 gives us True')
```

Example

```
print("Python Comparison Operators")
a = 5
b = 2
# equal to operator
print('a == b =', a == b)
# not equal to operator
print('a != b =', a != b)
# greater than the operator
print('a > b =', a > b)
# less than the operator
print('a < b =', a < b)
# greater than or equal to the operator
print('a >= b =', a >= b)
# less than or equal to the operator
print('a <= b =', a <= b)
```

Example

```
Print ("Python Logical Operators")
print ('and a and b Logical AND: True only if both the operands are True')
print ('or a or b Logical OR: True if at least one of the operands is True')
print ('not not a Logical NOT: True if the operand is False and vice-versa')
```

Example

```
print("Python Logical Operators")
# logical AND
print(True and True) # True
print(True and False) # False

# logical OR
print(True or False) # True

# logical NOT
print(not True) # False
```

Example

```
print("Python Bitwise operators")
print('&   Bitwise AND ')
print('|   Bitwise OR  ')
print('~   Bitwise NOT ')
print('^   Bitwise XOR ')
print('>>  Bitwise right')
print('<<  Bitwise left shift')
```

Example

```
number1=12345
number2=56789
print(bin(number1)[2:])
print(bin(number2)[2:])
number3=number1&number2
number4=number1|number2
number5=number1^number2
number6=~number1

print(bin(number3)[2:])
print(bin(number4)[2:])
print(bin(number5)[2:])
print(bin(number6)[3:])
number1 << 1
number2 >>=2
print(bin(number1)[3:])
print(bin(number2)[3:])
```


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')
```

output

```
b is greater than a
```

Python relies on indentation (whitespace at the beginning of a line)

Example

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

Output

```
File "D:\Pathon Programs\first semester program\cond1.py", line 4
```

```
    print("b is greater than a") # you will get an error
```

```
IndentationError: expected an indented block after 'if' statement on line 3
```

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")
```

Output

```
a and b are equal
```

Else

The `else` keyword catches anything which isn't caught by the previous 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")
```

Output

```
a is greater than b
```

Example

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
else:
    print("b is not greater than a")
```

Output

```
b is not greater than a
```

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:

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

Output

```
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")
```

Output

```
B
```

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")
```

Output

```
=
```

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")
```

Output

```
Both conditions are True
```

Or

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

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

Output

```
At least one of the conditions is True
```

Not

The `not` keyword is a logical operator, and is used to reverse the result of the conditional statement:

```
Example Test if a is NOT greater than b:
```

```
a = 33
b = 200
if not a > b:
    print("a is NOT greater than b")
```

Output

```
a is NOT greater than b
```

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.")
```

Output

```
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
```

Output

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
```

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
```

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)
```

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")
```

Python for Loops

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

Looping Through a String

Even strings are iterable objects, they contain a sequence of characters:

Example

Loop through the letters in the word "banana":

```
for x in "banana":  
    print(x)
```

The range() Function

To loop through a set of code a specified number of times, we can use the range() function,

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Example

Using the range() function:

```
for x in range(6):  
    print(x)
```

Note that range(6) is not the values of 0 to 6, but the values 0 to 5.

The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):

Example

Using the start parameter:

```
for x in range(2, 6):
    print(x)
```

The `range()` function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: `range(2, 30, 3)`

Example

Increment the sequence with 3 (default is 1):

```
for x in range(2, 30, 3):
    print(x)
```

Else in For Loop

The `else` keyword in a `for` loop specifies a block of code to be executed when the loop is finished:

Example

Print all numbers from 0 to 5, and print a message when the loop has ended:

```
for x in range(6):
    print(x)
else:
    print("Finally finished!")
```

Note: The `else` block will NOT be executed if the loop is stopped by a `break` statement.

Example

Break the loop when `x` is 3, and see what happens with the `else` block:

```
for x in range(6):
    if x == 3: break
    print(x)
else:
    print("Finally finished!")
```

Nested Loops

A nested loop is a loop inside a loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

Example

```
for i in range(3):
    for j in range(3):
        print(f'({i}, {j})')
```

This code will output:

```
(0,0)
(0,1)
(0,2)
(1,0)
(1,1)
(1,2)
(2,0)
(2,1)
(2,2)
```

The pass Statement

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

Example

```
for i in range(3):
    pass
```

Lecture 13

Example 1

- If the score is 90 or above, it assigns the grade 'A'.
- If the score is between 80 (inclusive) and 90 (exclusive), it assigns the grade 'B'.
- If the score is between 70 (inclusive) and 80 (exclusive), it assigns the grade 'C'.
- If the score is between 60 (inclusive) and 70 (exclusive), it assigns the grade 'D'.
- If none of the above conditions are met, it assigns the grade 'F'.

```
# Example of if-elif-else statement for grading
score = 75
if score >= 90:
    grade = 'A'
    print("grade",grade)
elif 80 <= score < 90:
    grade = 'B'
    print("grade",grade)
elif 70 <= score < 80:
    grade = 'C'
    print("grade",grade)
elif 60 <= score < 70:
    grade = 'D'
    print("grade",grade)
else:
    grade = 'F'
    print("grade",grade)
print(f"Your score is {score}, and your grade is {grade}.")
```

Example 2

```
# Example of if-elif-else statement to find the maximum of three numbers
num1 = 15
num2 = 24
num3 = 18
if num1 >= num2 and num1 >= num3:
    max_num = num1
elif num2 >= num1 and num2 >= num3:
    max_num = num2
else:
    max_num = num3

print(f"The maximum of {num1}, {num2}, and {num3} is {max_num}.")
```

- If `num1` is greater than or equal to both `num2` and `num3`, it assigns `max_num` the value of `num1`.
- If `num2` is greater than or equal to both `num1` and `num3`, it assigns `max_num` the value of `num2`.
- If neither of the above conditions is true (implying that `num3` is the greatest), it assigns `max_num` the value of `num3`.

Lecture 13

Example 3

```
# Example program to check if a number is positive, negative, or zero

# Taking user input
num = float(input("Enter a number: "))

# Checking if the number is positive, negative, or zero
if num > 0:
    print("The number is positive.")
elif num < 0:
    print("The number is negative.")
else:
    print("The number is zero.")
```

Example 4

```
# Example program to determine the eligibility for voting based on
age and citizenship

# Taking user input for age and citizenship
age = int(input("Enter your age: "))
citizenship = input("Are you a citizen? (yes/no): ").lower()

# Checking eligibility for voting
if age >= 18:
    if citizenship == "yes":
        print("You are eligible to vote. Exercise your right!")
    else:
        print("Sorry, you must be a citizen to vote.")
else:
    print("Sorry, you are not eligible to vote. You must be at least
18 years old.")
```

Example 5

```
original_string = "python"
reversed_string = original_string[::-1]

print("Original String:", original_string)
print("Reversed String:", reversed_string)
```

Output

```
Original String: python
Reversed String: nohtyp
```

Lecture 13

Example 6

```
# Taking user input for a word
user_word = input("Enter a word: ")

# Convert the word to lowercase for case-insensitive comparison
lowercase_word = user_word.lower()

# Check if the lowercase word is the same when reversed
if lowercase_word == lowercase_word[::-1]:
    print(f"{user_word} is a palindrome!")
else:
    print(f"{user_word} is not a palindrome.")
```

Note: In the context of palindrome checking, using `[::-1]` allows you to easily check if a word reads the same backwards as forward, as demonstrated in the previous examples.

Example 7

```
# Example of if-else statement to check if a number is even or odd
number = 42

if number % 2 == 0:
    print(f"The number {number} is even.")
else:
    print(f"The number {number} is odd.")
```

- If the number is divisible by 2 (i.e., the remainder when divided by 2 is 0), it prints that the number is even.
- If the remainder is not 0 (i.e., the number is not divisible by 2), it prints that the number is odd.

Lecture 13

Example 8

```
# Example of if-elif-else to categorize temperature
temperature = 28
if temperature < 0:
    category = "Freezing"
elif 0 <= temperature < 10:
    category = "Very Cold"
elif 10 <= temperature < 20:
    category = "Cold"
elif 20 <= temperature < 30:
    category = "Moderate"
elif 30 <= temperature < 40:
    category = "Warm"
else:
    category = "Hot"
print(f"The temperature is {temperature} degrees Celsius, which is
categorized as {category}.)")
```

Lecture 13

Example 9

```
# Taking user input for the number of sides

num_sides = int(input("Enter the number of sides of the geometric
shape: "))

# Checking the type of geometric shape based on the number of sides
if num_sides > 0:
    if num_sides == 3:
        print("It's a triangle.")
    elif num_sides == 4:
        print("It's a quadrilateral.")
    elif num_sides == 5:
        print("It's a pentagon.")
    elif num_sides == 6:
        print("It's a hexagon.")
    elif num_sides == 7:
        print("It's a heptagon.")
    elif num_sides == 8:
        print("It's an octagon.")
    else:
        print("It's a polygon with more than 8 sides.")
else:
    print("Please enter a valid number of sides.")
```