Python Programming

What is Python?[1]

Python is an interpreted high-level general-purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

[1] https://en.wikipedia.org/wiki/Python_(programming_language)

Who created Python?

Python was created by Guido van Rossum in the Netherlands in 1990. Van Rossum developed Python as a hobby, and Python has become a popular programming language widely used in industry and academia due to its simple, concise, and intuitive syntax and extensive library.

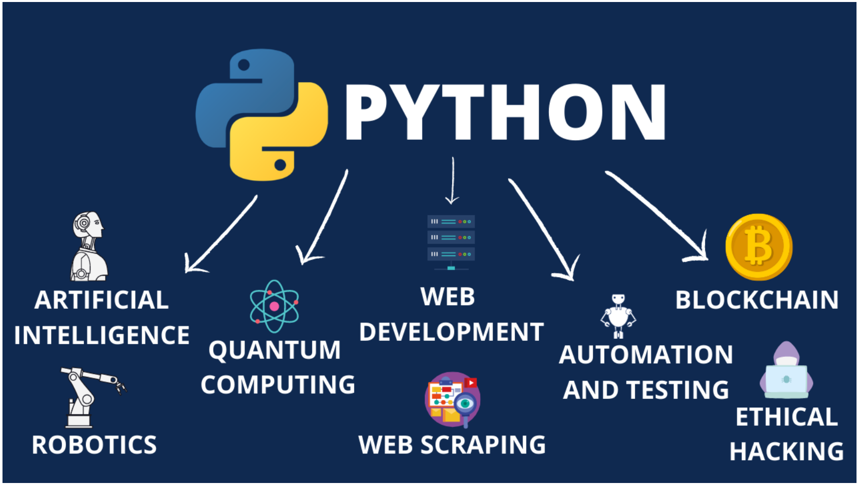
Why Python?

Because of many reasons[2]:

- **Easy to learn:** Learning Python is nothing but learning English.
- **Used everywhere:** Python is used in AI, ML(Artificial Intelligence and Machine Learning). It is also used in data science, games, apps, websites, automation, etc.
- **General purpose programming language:** It is a general purpose programming language which means it can be used on all devices and also in many different situations.
- **Loads of libraries:** Want to solve algebraic equations? Use numpy. Want to deal with images? Use PIL.
- **Just few lines of code does great:** With Python in just 5 lines of code is equivalent to, say, 20 lines of PHP code.
- **Great community:** Many people are willing to help with issues regarding Python. There is a great community.

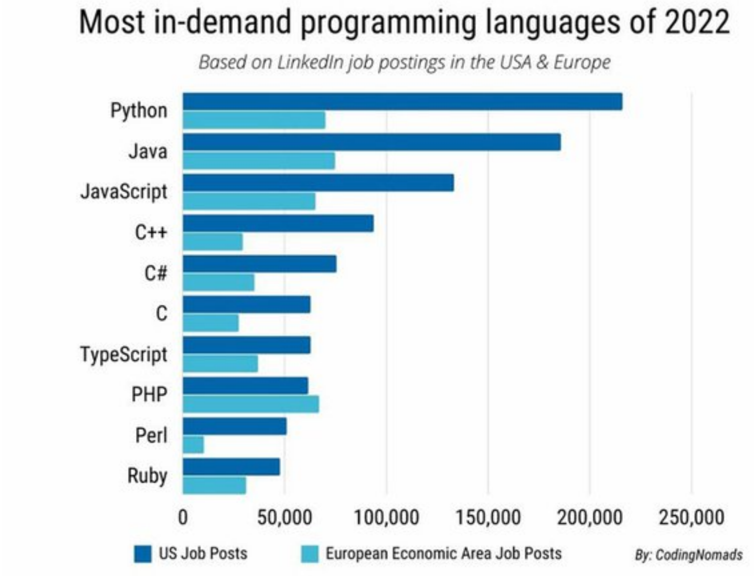
[2]https://www.quora.com/Why-do-people-still-use-Python

Why Python?[3]

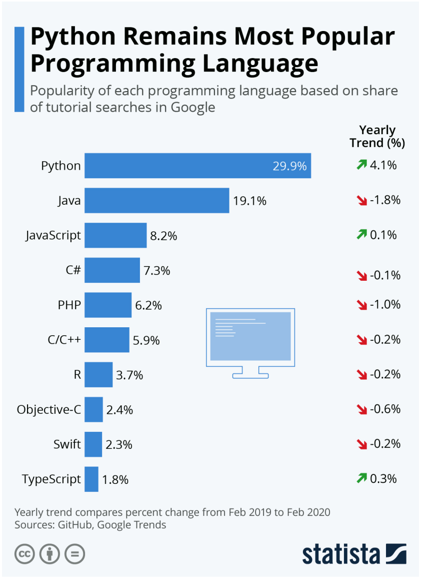


[3]https://becominghuman.ai/why-is-python-so-popular-b01a006b2be4

Why Python?


Most in-demand programming languages of 2022
Based on LinkedIn job postings in the USA & Europe
Python
Java
JavaScript
C++
C#
C
TypeScript
PHP
Perl
Ruby
0
50,000
100,000
150,000
200,000
250,000
US Job Posts
European Economic Area Job Posts
By: CodingNomads

Why Python?


Python Remains Most Popular Programming Language
Popularity of each programming language based on share of tutorial searches in Google
Yearly Trend (%)
Python 29.9% ↗4.1%
Java 19.1% ↘-1.8%
JavaScript 8.2% ↗0.1%
C# 7.3% ↘-0.1%
PHP 6.2% ↘-1.0%
C/C++ 5.9% ↘-0.2%
R 3.7% ↘-0.2%
Objective-C 2.4% ↘-0.6%
Swift 2.3% ↘-0.2%
TypeScript 1.8% ↗0.3%
Yearly trend compares percent change from Feb 2019 to Feb 2020
Sources: GitHub, Google Trends
statista

What is an open source/free software?[4]

Free and open-source software (FOSS) is software that is both free software and open-source software[a] where anyone is freely licensed to use, copy, study, and change the software in any way, and the source code is openly shared so that people are encouraged to voluntarily improve the design of the software.
Examples of free software: **VLC Media**, **Linux Mint**, **Gimp**, **Firefox**, ... etc.

[4]https://en.wikipedia.org/wiki/Free_and_open-source_software

What is a Licence?

A software license is a contract by which the copyright holder of a computer program defines with his co-contractor (operator or user) the conditions under which this program may be used, distributed or modified.

GNU Licence[5]

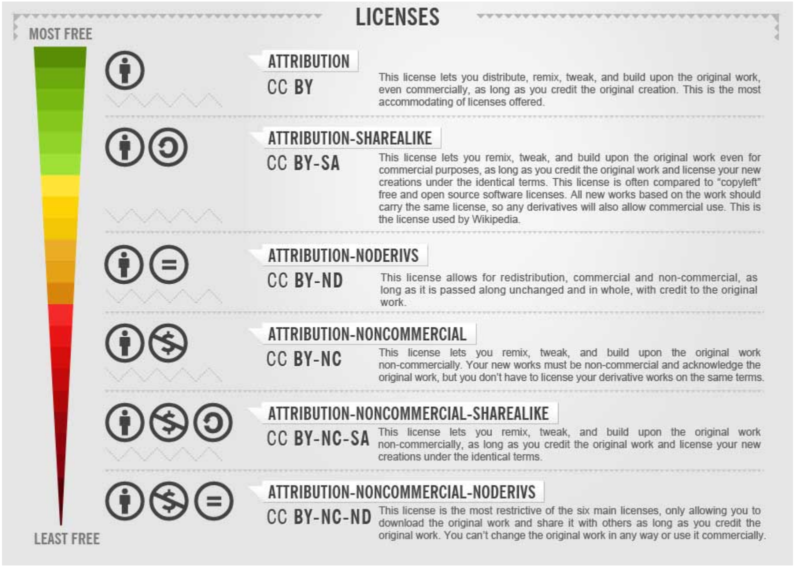
The GNU General Public License (GNU GPL or simply GPL) is a series of widely used free software licenses that guarantee end users the four freedoms to run, study, share, and modify the software.

GPL 3 Free Software

Free as in Freedom

[5] https://en.wikipedia.org/wiki/GNU_General_Public_License

Levels of freedom of a software



LICENSES

MOST FREE

ATTRIBUTION

CC **BY**

This license lets you distribute, remix, tweak, and build upon the original work, even commercially, as long as you credit the original creation. This is the most accommodating of licenses offered.

ATTRIBUTION-SHAREALIKE

CC **BY-SA**

This license lets you remix, tweak, and build upon the original work even for commercial purposes, as long as you credit the original work and license your new creations under the identical terms. This license is often compared to "copyleft" free and open source software licenses. All new works based on the work should carry the same license, so any derivatives will also allow commercial use. This is the license used by Wikipedia.

ATTRIBUTION-NODERIVS

CC **BY-ND**

This license allows for redistribution, commercial and non-commercial, as long as it is passed along unchanged and in whole, with credit to the original work.

ATTRIBUTION-NONCOMMERCIAL

CC **BY-NC**

This license lets you remix, tweak, and build upon the original work non-commercially. Your new works must be non-commercial and acknowledge the original work, but you don't have to license your derivative works on the same terms.

ATTRIBUTION-NONCOMMERCIAL-SHAREALIKE

CC **BY-NC-SA**

This license lets you remix, tweak, and build upon the original work non-commercially, as long as you credit the original work and license your new creations under the identical terms.

ATTRIBUTION-NONCOMMERCIAL-NODERIVS

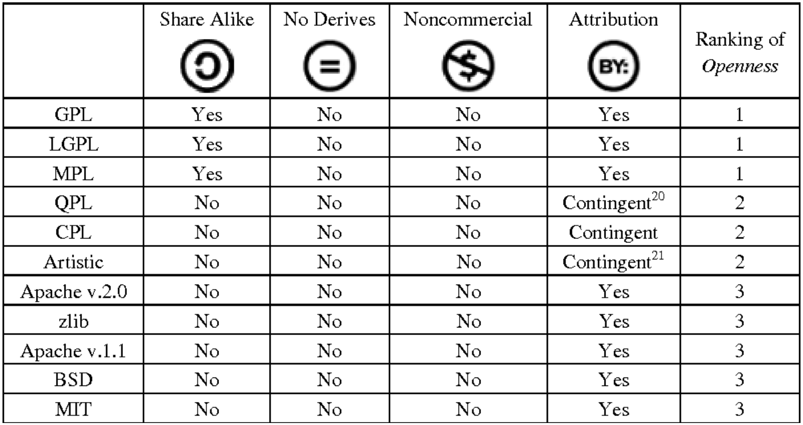
CC **BY-NC-ND**

This license is the most restrictive of the six main licenses, only allowing you to download the original work and share it with others as long as you credit the original work. You can't change the original work in any way or use it commercially.

LEAST FREE

Is there other types of licences?

Table 2. Ranking of FOSS licenses' degree of *Openness* based on CC elements.



	Share Alike	No Derives	Noncommercial	Attribution	Ranking of *Openness*
GPL	Yes	No	No	Yes	1
LGPL	Yes	No	No	Yes	1
MPL	Yes	No	No	Yes	1
QPL	No	No	No	Contingent[20]	2
CPL	No	No	No	Contingent	2
Artistic	No	No	No	Contingent[21]	2
Apache v.2.0	No	No	No	Yes	3
zlib	No	No	No	Yes	3
Apache v.1.1	No	No	No	Yes	3
BSD	No	No	No	Yes	3
MIT	No	No	No	Yes	3

What we will study during the semester

Chap 1	Introduction to computers, Programs and Python
Chap 2	Elementary Programming
Chap 3	Mathematical Functions and Strings
Chap 4	Conditions
Chap 5	Loops

What we will study during the semester

Chap 6	Functions
Chap 7	Lists
Chap 8	Tuples, Sets and Dictionaries
Chap 9	Multidimensional Lists
Chap 10	Files and Exceptions Handling

Differents ways to run Python

- Interactive Mode (Command prompt).
- Command Line (File running) 'python file.py'.
- Text Editor (VS Code, Jupyter).
- IDE (PyCharm)

Type of Errors

- **Syntax errors** result from errors in code construction, such as mistyping a statement, incor- rect indentation, omitting some necessary punctuation, or using an opening parenthesis with- out a corresponding closing parenthesis.
- **Runtime errors** are errors that cause a program to terminate abnormally. They occur while a program is running if the Python interpreter detects an operation that is impossible to carry out.
- **Logic errors** occur when a program does not perform the way it was intended to. Errors of this kind occur for many different reasons.

Objective of the course

At the end of this semester, you will be able to solve a problem and write its code in Python.

Elementary programming

Comments

All modern programming languages have comment characters. These indicate part of the code that should be skipped by the interpreter.

Why 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.

How we write comments ?

Any characters after a sharp (#) on a line are skipped.

Elementary programming

```
#This is a comment
#written in
#more than just one line
print("Hello, NHSM!")
#print("This line will be ignored ")
print("Did you understand comments ?") #This line will print "Did ...?"
```

Python multiline comments

Python doesn't support multiline comments.
However, you can use two triple quotes (""") as multiline comments.
Guido van Rossum, the creator of Python, also recommended this.

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

Variables assignement

Variables consist of two parts: the identifier (name) and the value.

To assign a variable to a name, use a single equals sign (=).
A variable is created the moment you first assign a value to it.

```
Name = "Omar" #Create a variable named 'Name' with String value 'Omar'
Age = 5 #Create a variable named 'Age' with Integer value '25'
High = 1.73 #Create a variable named 'Age' with real value '1.73'
```

Python allows you to assign a single or multiple values (not mandatory of the same type) to several variables simultaneously.

```
a = b = c = 10 #assign a single 10 to a,b and c
Name,Age,High = "Omar",2,1.73 #assign 3 different values to 3 variables.
```

Elementary programming

Type of variable

You don’t do this in Python, however, because Python automatically figures out the data type according to the value assigned to the variable.
The most used types in Python are:

```
int #numerical type
float #numerical type
str #for textual manipulation
bool #logical type
complex #numerical type
```

You can determine the type of a variable or a literal value by using the type() function.

```
>>> High = 1.73
>>> type(High)
<class 'float'>
>>> type(43)
<class 'int'>
```

Elementary programming

type conversion

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

```
x = 5 # int
y = 1.81 # float
z = 3+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))
```

Dynamic types of variables

Python is dynamically typed. This means that:

- Types are set on the variable values and not on the variable names.
- Variable types do not need to be known before the variables are used.
- Variable names can change types when their values are changed.

```
a=3
a="yes"
a=1.73
```

Naming variables

- Variable names can contain only letters, numbers, and underscores `(_)`. They can start with a letter or an underscore `(_)`, not with a number.
- Variable names cannot contain spaces. To separate words in variables, you use underscores for example `sorted_list`
- Variable names cannot the same as keywords, reserved words, and built-in functions in Python.

The following guidelines help you define good variable names:

- Variable names should be concise and descriptive. For example, the `active_user` variable is more descriptive than the `au`.
- Use underscores `(_)` to separate multiple words in the variable names.
- Avoid using the letter `l` and the uppercase letter `O` because they look like the number `1` and `0`.

Operators & Evaluating expressions

Operators are the syntax that Python uses to express common ways to manipulate data and variables.

Python divides the operators in the following groups:

Arithmetic operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Usage
`+`	Addition	`x + y`
`-`	Subtraction	`x - y`
`*`	Multiplication	`x * y`
`/`	Float Division	`x / y`
`%`	Remainder	`x % y`
`**`	Exponentiation	`x ** y`
`//`	Integer division	`x // y`

Operators & Evaluating expressions

Assignment operators

Assignment operators are used to assign values to variables:

Operator	Name	Usage
`=`	assignment	`x=3`
`+=`	addition assignment	`x+=3`
`-=`	subtraction assignment	`x-=3`
`*=`	multiplication assignment	`x*=3`
`/=`	float division assignment	`x/=3`
`//=`	integer division assignment	`x//=3`
`%=`	remainder assignment	`x%=3`
`**=`	exponent assignment	`x**=3`

Operators & Evaluating expressions

Relational operators

Logical operators are used to combine conditional statements:

Operator	Name	Usage
`<`	less than	`x<y`
`<=`	less than or equal to	`x<=y`
`>`	greater than	`x>y`
`>=`	greater than or equal to	`x>=y`
`==`	equal to	`x==y`
`!=`	not equal	`x!=y`

Operators & Evaluating expressions

Logical operators

Logical operators are used to combine conditional statements:

Operator	Name	Usage
`and`	logical conjunction	`x < 5 and  x < 10`
`or`	logical disjunction	`x < 5 or x < 4`
`not`	logical negation	`not(x < 5 and x < 10)`

Python Operators Precedence Rule

- An expression is made with combinations of variables, values, operators and function calls.
- The Python interpreter evaluates the valid expression.
- Python uses a type of rule known as **PEMDAS**.
  1. **P**: Parentheses
  2. **E**: Exponentiation
  3. **M**: Multiplication
  4. **D**: Division
  5. **A**: Addition
  6. **S**: Subtraction
- Operators with the same precedence (except for `**`) are evaluated from left-to-right.

Input / output data

```
Age=eval(input("How old are you ?"))# input statement
print(Age)#Output (display) statement
```

Types of Errors

Syntax Errors

Syntax errors are the most basic type of error. They arise when the Python parser is unable to understand a line of code. Syntax errors are almost always fatal, i.e. there is almost never a way to successfully execute a piece of code containing syntax errors

Runtime Errors

A program with a runtime error is one that passed the interpreter's syntax checks, and started to execute. However, during the execution of one of the statements in the program, an error occurred that caused the interpreter to stop executing the program and display an error message. Runtime errors are also called exceptions because they usually indicate that something exceptional (and bad) has happened. Some examples of Python runtime errors:

1. division by zero
2. performing an operation on incompatible types
3. using an identifier which has not been defined
4. accessing a list element, dictionary value or object attribute which doesn't exist
5. trying to access a file which doesn't exist

Logic Errors

These are the most difficult type of error to find, because they will give unpredictable results and may crash your program. A lot of different things can happen if you have a logic error. However these are very easy to fix as you can use a debugger, which will run through the program and fix any problems.
Here are some examples of mistakes which lead to logical errors:

1. using the wrong variable name
2. indenting a block to the wrong level
3. using integer division instead of floating point division
4. getting operator precedence wrong
5. making a mistake in a boolean expression
6. off-by-one, and other numerical errors

Elementary programming

Documentation

There is online and offline versions of Python documentation

- online: `https://www.python.org/doc/`
- offline: Zeal `https://zealdocs.org/`

How to find a solution for a problem

- Write the Errors on Google as it is.
- Ask a question on a forum.

Math functions and Strings

Mathematical functions

Python has a limited number of **built-in mathematical functions**:

```
abs( number ) #Return the absolute value of the argument, |x|.
pow( x , y , [ z ]) #Raise x to the y power. If z is present, this is done
                    #modulo z , x y % z .
round( number , [ ndigits ]) #Round number to ndigits beyond the decimal
                             #point (rounds a number to the nearest
                             #whole number).
cmp( x , y ) #Compare x and y , returning a number.
hex( number )#Create a hexadecimal string representation of number . A
             #leading '0x' is placed on the string as a reminder that
             #this is hexadecimal.
oct( number )#Create a octal string representation of number .
             #A leading '0' is placed on the string as a reminder that this
             #is octal not decimal.
int( string , [ base ]) #Generates an integer from the string x . If base is
                        #supplied, x must be in the given base. If base is
                        #omitted, x must be decimal.
max( sequence ) #Return the largest value in sequence .
min( sequence ) #Return the smallest value in sequence .
ord( character ) #Returns the Unicode code from a given character.
chr( character ) #Returns a character whose Unicode code point is an integer
```

Math functions and Strings

math module functions

More advanced mathematical functions are contained in the `math` **module**.

A **module** is a file that contains a collection of related functions.

Before we can use the functions from a module, we have to import them:

```
import math # call module before use
angle = 90 * 2 * math.pi / 360.0
math.sin(angle)
```

Math functions and Strings

Examples

```
math.ceil(x) #Returns the smallest integer greater than or equal to x.
math.fabs(x) #Returns the absolute value of x
math.factorial(x) #Returns the factorial of x
math.floor(x) #Returns the largest integer less than or equal to x
math.fmod(x, y) #Returns the remainder when x is divided by y
math.exp(x) #Returns e**x
math.log(x[, b]) #Returns the logarithm of x to the base b (defaults to e)
math.log2(x) #Returns the base-2 logarithm of x
math.log10(x) #Returns the base-10 logarithm of x
math.pow(x, y) #Returns x raised to the power y
math.sqrt(x) #Returns the square root of x
math.acos(x) #Returns the arc cosine of x
math.asin(x) #Returns the arc sine of x
math.atan(x) #Returns the arc tangent of x
```

Examples

```
math.cos(x) #Returns the cosine of x
math.sin(x) #Returns the sine of x
math.tan(x) #Returns the tangent of x
math.degrees(x) #Converts angle x from radians to degrees
math.radians(x) #Converts angle x from degrees to radians
math.acosh(x) #Returns the inverse hyperbolic cosine of x
math.asinh(x) #Returns the inverse hyperbolic sine of x
math.atanh(x) #Returns the inverse hyperbolic tangent of x
math.cosh(x) #Returns the hyperbolic cosine of x
math.sinh(x) #Returns the hyperbolic cosine of x
math.tanh(x) #Returns the hyperbolic tangent of x
math.pi #Mathematical constant, the ratio of circumference of a
        #circle to it's diameter (3.14159...)
math.e #Mathematical constant e (2.71828...)
```

Math functions and Strings

What is String in Python?

A string is a sequence of characters.

How to create a string in Python?

Strings can be created by enclosing characters inside a single quote or double-quotes.
Even triple quotes can be used in Python but generally used to represent multiline strings and docstrings.

```
# defining strings in Python, all of the following are equivalent
say_hello = 'Hello'

say_hello = "Hello"

say_hello = '''Hello'''

# triple quotes string can extend multiple lines
say_hello = """Hello, welcome to
           the world of Python"""
```

How to access characters in a string?

- We can access individual characters using **indexing** and a range of characters using **slicing**.
- Python is zero-indexing language, Index starts from 0.
- Trying to access a character out of index range will raise an `IndexError`.
- The index must be an integer. We can't use floats or other types, this will result into `TypeError`.
- Python allows negative indexing for its sequences.
- The index of -1 refers to the last item, -2 to the second last item and so on. We can access a range of items in a string by using the slicing operator :(colon).

Math functions and Strings

```
#Accessing string characters in Python
my_word = 'ThisIsALongWord'
print('my_word = ', my_word)

#first character
print('my_word[0] = ', my_word[0])

#last character
print('my_word[-1] = ', my_word[-1])

#slicing 2nd to 5th character
print('my_word[1:5] = ', my_word[1:5])

#slicing 6th to 2nd last character
print('my_word[5:-2] = ', my_word[5:-2])
```

Now, we try to access an index out of the range or use numbers other than an integer

```
# index must be in range
my_word[16]
# index must be an integer
my_word[1+0j]
```

Common Operations on Strings

Concatenation of Two or More Strings

Joining of two or more strings into a single one is called concatenation. The + operator does this in Python. Simply writing two string literals together also concatenates them.
The * operator can be used to repeat a string for a given number of times.

```
# Python String Operations
str1 = 'Hello'
str2 ='NHSM!'

# using +
print('str1 + str2 = ', str1 + str2)
# using *
print('str2 * 3 =', str2 * 3)
# two string literals together
str1='Hello ''World!'
print('str1 =', str1)
```

Math functions and Strings

String Membership Test

We can test if a substring exists within a string or not, using the keyword `in`

```
'me' in 'home'
'ho' not in 'home'
```

Built-in functions to Work with Python

The `enumerate()` function returns an enumerate object.
The `len()` returns the length (number of characters) of the string.

```
word="Hello NHSM"
print(enumerate(word))
print(len(word))
```

Common String methods

Some of the commonly used methods are `lower()`, `upper()`, `join()`, `split()`, `find()`, `replace()` etc.

```
word="PythonIsNotFunny"
print(word.lower())
print(word.upper())
sentence="This will split all words into a list"
print(sentence.split())
sentence="This will split, all words, into a list"
print(sentence.split(','))
word_list=['This', 'will', 'join', 'all', 'words', 'into', 'a', 'string']
delimiter="*"
print(delimiter.join(word_list))
print(' '.join(word_list))
print('Happy Day with python'.find('ay'))
print('Happy Day with python'.replace('Happy','Brilliant'))
```

Indentation

Python Indentation

Indentation is a very important concept of Python because without proper indenting the Python code, you will end up seeing IndentationError and the code will not get compiled.
In simple terms indentation refers to adding white space before a statement.

```
Username = 'Omar_213'
if Username == 'Omar_213':
    print('Logging on to website...')
    password=input('Enter your password')
else:
    print('You are in the wrong place.')
print('All set !')
```

Python Indentation Rules

Python uses 4 spaces as default indentation spaces. However, the number of spaces can be anything, it is up to the user. But a minimum of one space is needed to indent a statement.

- The first line of python code cannot have Indentation.
- Indentation is mandatory in python to define the blocks of statements.
- The number of spaces must be uniform in a block of code.
- It is preferred to use whitespaces instead of tabs to indent in python. Also, either use whitespace or tabs to indent, intermixing of tabs and whitespaces in indentation can cause wrong indentation errors.

Indentation

Benefits of Indentation in Python

- Indentation of code leads to better readability, although the primary reason for indentation in python is to identify block structures.
- Missing `{` and `}` errors that sometimes popup in `c,c++` languages (or other languages) can be avoided in python, also the number of lines of code is reduced.

Conditions

if statement

Some programming problems need special treatment.
Let consider the following situation:

Write a program that asks the user's age, then prints out the message "You are still young!" if his age is less than 30.

to solve this problem, we need to use `if` statement given by the following syntax:

```
if (test): #condition
    block of instructions #body
```

- An `if` statement executes the statements if the condition is true, and ignores it if the condition is false.
- The block of instructions must starts with and indentation and will end at the first unindented statement.

Conditions

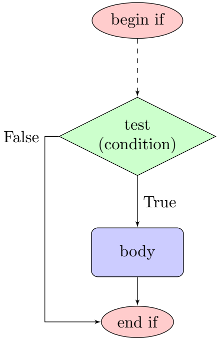


Figure 1: Python `if` Statement Flowchart

Therefore, the solution of the previous situation is:

```
age=int(input("How old are you"))
if (age<30):
      print("You are still young!")
```

Conditions

`if...else` statements

If we have two outputs based on whether the condition is true or false, we use the `if...else` statements

- When the condition evaluates to `True` executes the `body_of_if` and the `body_of_else` is skipped.
- When the condition evaluates to `False` the `body_of_if` is skipped and the `body_of_else` executes.

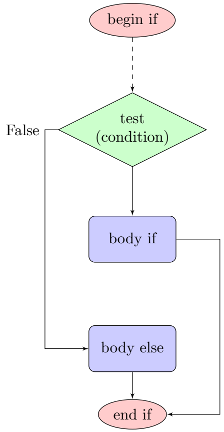


Figure 2: `if..else` Flowchart

Conditions

The syntax of `if...else` statement is:

```
if conditional_test:
      body_of_if
else:
      body_of_else
```

Example

Write a program that asks the user to enter two numbers, then prints out the max between them without using any built-in function or module.

```
a,b=int(input("Give the first number")),int(input("Give the second number"))
if (a>=b):
      print(a)
else:
      print(b)
```

Conditions

Example

Let suppose that we want to develop a quiz to practice subtraction for kids. Then
Write a program which do the following steps:

- Generate two random numbers, `num1` and `num2` between 0 and 100 (use the function `randint(0,100)` from the module `random`).
- If `(num1<num2)`, swap the numbers `num1` and `num2`.
- Asks the user to enter the result of the subtraction by print out ("What is ",`num1`,"-",`num2`).
- Check the user's answer and display whatever the answer is correct or no.

Conditions

if..elif..else statement

Often, we are faced the case of more than two possible results of the conditional test, to evaluate these we can use the `if-elif-else` statement syntax:

```
#Chained conditional

if conditional_test1:
    body_of_if
elif conditional_test2:
    body_of_elif
elif conditional_test3:
    body_of_elif
       .
       .
       .
else:
    body_of_else
```

$\Longleftrightarrow$

```
if conditional_test1:
    body_of_if
else:
    if conditional_test2:
        body_of_else_if
    else:
        if conditional_test3:
            body_of_else_if
                .
                  .
                    . #Nested conditional
                    else:
                        body_of_else
```

- In the right side statement, when we place an `if` statement inside another, we form a **nested if statement**.
- In python, the chained conditional are better then nested.

Conditions

Example

Write a program which check if given number is positive, Zero or negative.

```
#Chained conditional
num=int(input("Enter a number\n"))
if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

```
#Nested
num=int(input("Enter a number\n"))
if num > 0:
    print("Positive number")
else:
    if num == 0:
        print("Zero")
    else:
        print("Negative number")
```

Common errors in conditions

- The operator of comparison is `==` and not `=`:

```
if (a==0):#correct
if (a=0):#incorrect
```

- Use `and` where `or` is or vice-versa carefully. For example, which conditional tests is true if $1 \leq x \leq 100$?

```
if x>1 and x<100: #Correct
if x>1 or x<100: #Incorrect
```

- The common mistake below will generate `SyntaxError`.

```
if age>1 and age<100: #Correct
if age>1 and <100: #Incorrect
```

- Indentation errors are the most common errors in conditions.

```
if age>1 and age<100:
    age+=2
        print(age)#Must be in the same column as the first instruction
```

Conditions

Exercise 1:

In algebra, a quadratic equation is an equation in the form of

$$ax^2 + bx + c = 0$$

Write a program to find all roots (real or complex) of a quadratic equation using `if else`.

Loops

Loops

Exercise 2:

Write a program which print the following messages
"Hi Student 1"
"Hi Student 2"
⋮
"Hi Student 90"
"Hi Student 100"

Will you print all the messages using 100 `print` function?!! So What we can do ?

In programming, loops are the statements that allow executing the same block of instructions multiple times.
Python has essentially two types of loops: `while loops` and `for loops`.

While loops

- A `while` loop allows a block of instructions to execute as long as (or while) a condition is `True`.
- When the condition is `False`, the loop's body is ignored, and the execution continues at the first statement after the body of the `while` loop.
- The body of the while loop is determined through indentation.
- If the condition always evaluates to `True`, the body will be executed infinitely, that's what we call **infinite loop problem**.
- We must update the condition's variables to avoid the infinite loop.
- Generally, we use while loops when we don't know the number of executions of the loop's body.

Loops

while syntax and flowchart

while's loop syntax is:

```
while (condition):
    block_instructions #body
```

Listing 1: while syntax

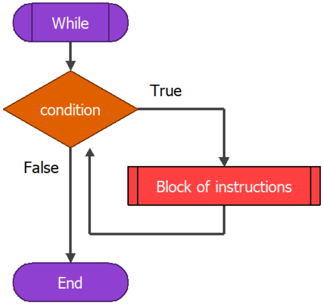


Figure 3: while flowchart

Loops

While loop with else

`while` loop can have an optional else. The else body is executed if the condition is `False`

```python
counter = 0
while counter < 3:
    print("Inside while body")
    counter = counter + 1
else:
    print("Inside else body")
```

for loops

- The `for` loop allows to execute a block of instructions multiple times. In fact, this loop is used to iterate over iterable objects.
- An **iterable objects** is a sequence of items capable of returning its members one by one, for example `list` and `strings` are iterable objects.

The `for` loop syntax in python is:

```
for loop_var in iterable_object:
    for_block #body
```

The variable `loop_var` take the values of the items of the iterable object. Loop continues until the variable reach the last item in the sequence.
The body of the loop must be indented.

Loops

Exercise 3:

Draw the flowchart of for loop

Example: Python `for` Loop

```
word="NHSM"
for letter in word:
      print(letter)
```

Loops

range() function

In python, we can use the range() function to generate a sequence of numbers. The usage of range function is as follows:

```
range([start], stop,[step_size])
```

`start` :(Optional). An integer number specifying at which position to start. Default is 0.

`stop` :An integer number specifying at which position to stop (not included).

`step` :(Optional). An integer number specifying the incrementation. Default is 0.

Try this

```
print(range(10))
print(list(range(10)))
print(list(range(2, 8)))
print(list(range(2, 20, 3)))
```

Examples

```
#example1
for i in range(9):
    print(i)
else:
    print("No items left.")
#example2 (else)
digits=[6,2,9]
for digit in digits:
    print(digit)
else:
    print("No items left.")
#example3
# Program to iterate through a list using indexing
names = ['Omar', 'Mohamed', 'Ali']
# iterate over the list using index
for i in range(len(names)):
    print("His name is", names[i])
```

Loops

Exercise 4:

Write a program which print the following messages
"Hi Student 1"
"Hi Student 2"
⋮
"Hi Student 90"
"Hi Student 100"

Will you print all the messages using 100 `print` function?!! So What we can do ?

In programming, loops are the statements that allow executing the same block of instructions multiple times.
Python has essentially two types of loops: `while loops` and `for loops`.

Loops

`range()` function

In python, we can use the range() function to generate a sequence of numbers. The usage of range function is as follows:

```
range([start], stop,[step_size])
```

`start` :(Optional). An integer number specifying at which position to start. Default is 0.

`stop` :An integer number specifying at which position to stop (not included).

`step` :(Optional). An integer number specifying the incrementation. Default is 0.

Try this

```python
print(range(10))
print(list(range(10)))
print(list(range(2, 8)))
print(list(range(2, 20, 3)))
```

Examples

```
#example1
for i in range(9):
    print(i)
else:
    print("No items left.")
#example2 (else)
digits=[6,2,9]
for digit in digits:
    print(digit)
else:
    print("No items left.")
#example3
# Program to iterate through a list using indexing
names = ['Omar', 'Mohamed', 'Ali']
# iterate over the list using index
for i in range(len(names)):
    print("His name is", names[i])
```

break and continue

- The **break** statement terminates the loop containing it. The program execute the statement immediately after the body of the loop.
- The **continue** statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.

```
name="NHSM"
for letter in name:
    if (letter=='S'):
        break
    print(letter)
```

```
    name="NHSM"
    for letter in name:
        if (letter=='S'):
            continue
        print(letter)
```