V-Ex Tech

V-EX TECH

Web Development

Java / Node.js / PHP / .Net / Python

Certification Course

Assured Placement Program

With International Certificate

About V-Ex Tech….

V-Ex Tech is an elevated education platform providing rigorous industry-relevant programs Designed and delivered on collaboration with industry professionals. It has been constantly Into process of creating an immersive learning experience binding latest technologies, pedagogy and services with enormous job placement opportunities too.

V-Ex Tech

PYTHON

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

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.

C LANGUAGE

```
#include <stdio.h>

int main() {
  printf("Hello World!");
  return 0;
}
```

Python File Handling

V-Ex Tech

Python Database Handling

INSTALLATION

```
C:\Users\python -version
```

https://www.python.org/downloads/

Python Syntax

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

Python Indentation

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

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

Comments

```
#This is a comment.

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

Python Variables

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

Get the Type

```
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:

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

Case-Sensitive

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

Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). 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)

Allowed

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

NOT ALLOWED

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

Many Values to Multiple Variables

```
x, y, z = "Orange", "Banana", "Cherry"

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

One Value to Multiple Variables

```
x = y = z = "Orange"

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

Output Variables

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

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

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

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

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

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

Global Variables

```
x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()
```

```
x = "awesome"

def myfunc():
  x = "fantastic"
  print("Python is " + x)

myfunc()

print("Python is " + x)
```

```
def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

Python Data Types

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`

V-Ex Tech

Python Numbers

- int
- float
- complex

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

Python Casting

x = int(1)

y = int(2.8)

z = int("3")

print(x)

print(y)

print(z)

Python Strings

'hello' is the same as "hello".

Multiline Strings

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

Slicing Strings

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

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

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

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

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

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

Modify Strings

Upper Case

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

Lower Case

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

Remove Whitespace

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

Replace String

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

Split String

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

String Format

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

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

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

```
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))
```

Escape Character

```
txt = "We are the so-called "Vikings" from the north."
```

```
txt = "We are the so-called \"Vikings\" from the north."
```

String method: https://www.w3schools.com/python/python_strings_methods.asp

Python Booleans

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

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

```
a = 200
b = 33

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

Any string is `True`, except empty strings.

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

V-Ex Tech

Python Operators

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

Python Arithmetic Operators

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

Python Assignment Operators

Operator	Example	Same As	Try it
=	x = 5	x = 5	Try it »
+=	x += 3	x = x + 3	Try it »

-=	x -= 3	x = x - 3	Try it »
*=	x *= 3	x = x * 3	Try it »
/=	x /= 3	x = x / 3	Try it »
%=	x %= 3	x = x % 3	Try it »

Python Comparison Operators

==	Equal	x == y	Try i
!=	Not equal	x != y	Try i
>	Greater than	x > y	Try i

<	Less than	x < y	Try i
>=	Greater than or equal to	x >= y	Try i
<=	Less than or equal to	x <= y	

```
x = 5
y = 3

print(x != y)

# returns True because 5 is not equal to 3
```

Python Logical Operators

```
x = 5

print(x > 3 and x < 10)
```

```
x = 5

print(x > 3 or x < 4)
```

```
x = 5
print(not(x > 3 and x < 10))
```

Python Membership Operators

```
x = ["apple", "banana"]

print("banana" in x)
```

```
x = ["apple", "banana"]

print("pineapple" not in x)
```

Operator Precedence

```
print((6 + 3) - (6 + 3))
```

Python Lists

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

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

Lists are created using square brackets:

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

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.

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

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

List items can be of any data type:

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

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

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

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.

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

```
thislist =
["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

change data

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

```
thislist[1:3] = ["watermelon"]
```

```
print(thislist)
```

Add List Items

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

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

tropical = ["mango", "pineapple", "papaya"]

thislist.extend(tropical)

print(thislist)
```

Remove List Items

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

Remove Specified Index

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

```
thislist = ["apple", "banana", "cherry"]
del thislist
```

Clear the List

The `clear()` method empties the list.

The list still remains, but it has no content.

Example

Clear the list content:

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

Loop Through a List

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

Loop Through the Index Numbers

Use the `range()` and `len()` functions

```
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
  print(thislist[i])
```

Looping Using List Comprehension

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

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
  if "a" in x:
    newlist.append(x)

print(newlist)
```

Comprehension

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)
```

The Syntax

```
newlist = [expression for item in iterable if condition == True]
```

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if x != "apple"]

print(newlist)
```

Sort Lists

Sort List Alphanumerically

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
```

```
thislist.sort()


print(thislist)
```

Sort Descending

To sort descending, use the keyword argument `reverse = True`:

Example

Sort the list descending:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort(reverse = True)
print(thislist)
```

Case Insensitive Sort

By default the `sort()` method is case sensitive, resulting in all capital letters being sorted before lower case letters:

Example

Case sensitive sorting can give an unexpected result:

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort()
print(thislist)
```

Reverse Order

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.reverse()
print(thislist)
```

Copy a List

You cannot copy a list simply by typing `list2 = list1`, because: `list2` will only be a *reference* to `list1`, and changes made in `list1` will automatically also be made in `list2`.

There are ways to make a copy, one way is to use the built-in List method `copy()`

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

Join Two Lists

```
list1 = ["a", "b", "c"]

list2 = [1, 2, 3]


list3 = list1 + list2

print(list3)
```

```
list1 = ["a", "b" , "c"]
```

```
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)

print(list1)
```

List count() Method

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

x = fruits.count("cherry")
```

append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value

extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	

Python Tuples

```
mytuple = ("apple", "banana", "cherry")
```

A tuple is a collection which is ordered and **unchangeable**.

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

Unchangeable

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

Allow Duplicates

```
thistuple = ("apple", "banana", "cherry", "apple", "cherry")
print(thistuple)
```

Tuple Length

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

Create Tuple With One Item

To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

```
thistuple = ("apple",)

print(type(thistuple))


#NOT a tuple

thistuple = ("apple")

print(type(thistuple))
```

Data Types

```
tuple1 = ("abc", 34, True, 40, "male")


thistuple = ("apple", "banana", "cherry")
print(thistuple[1])


thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])


thistuple =
("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])
```

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

Update Tuples

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)
```

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.append("orange")
thistuple = tuple(y)
```

Remove Items

Tuples are **unchangeable**, so you cannot remove items from it, but you can use the same workaround as we used for changing and adding tuple items:

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
thistuple = tuple(y)
```

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple no longer
exists
```

unpack tuple

```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")


(green, yellow, *red) = fruits


print(green)

print(yellow)

print(red)
```

Loop Through a Tuple

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

```
thistuple = ("apple", "banana", "cherry")
for i in range(len(thistuple)):
  print(thistuple[i])
```

Join Two Tuples

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

Multiply Tuples

```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)
```

V-Ex Tech

PYTHON TEST

1)How To Find the Length of a List in Python

```
lst = [10,20,30,40]
```

2)

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

a)orl

b)Worl

c)orld

3)Print 100 number using loop in list ,number will start from0

4) thistuple = (0, "banana", "cherry",”mango”,7,4)

Remove 3rd index and print it

Python Sets

```
myset = {"apple", "banana", "cherry"}
```

A set is a collection which is *unordered*, *unchangeable*

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

Duplicates Not Allowed

```
thisset = {"apple", "banana", "cherry", "apple"}

print(thisset)
```

```
thisset = {"apple", "banana", "cherry", True, 1, 2}

print(thisset)
```

Get the Length of a Set

```
thisset = {"apple", "banana", "cherry"}

print(len(thisset))
```

Set Items - Data Types

```
set1 = {"abc", 34, True, 40, "male"}
```

```
myset = {"apple", "banana", "cherry"}
print(type(myset))
```

Once a set is created, you cannot change its items, but you can add new items.

```
thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)
```

```
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}

thisset.update(tropical)

print(thisset)
```

```
thisset = {"apple", "banana", "cherry"}
mylist = ["kiwi", "orange"]

thisset.update(mylist)

print(thisset)
```

Remove Set Items

```
thisset = {"apple", "banana", "cherry"}

thisset.remove("banana")

print(thisset)
```

Remove a random item by using the `pop()` method:

```
thisset = {"apple", "banana", "cherry"}

x = thisset.pop()

print(x)

print(thisset)
```

```
thisset = {"apple", "banana", "cherry"}

thisset.clear()

print(thisset)
```

The `del` keyword will delete the set completely:

```
thisset = {"apple", "banana", "cherry"}

del thisset

print(thisset)
```

Loop Sets

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

Join Sets

You can use the `union()` method that returns a new set containing all items from both sets, or the `update()` method that inserts all the items from one set into another:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)
```

The `update()` method inserts the items in set2 into set1:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set1.update(set2)
print(set1)
```

Keep ONLY the Duplicates

The `intersection_update()` method will keep only the items that are present in both sets.

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.intersection_update(y)

print(x)
```

Keep All, But NOT the Duplicates

The `symmetric_difference_update()` method will keep only the elements that are NOT present in both sets.

Example

Keep the items that are not present in both sets:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.symmetric_difference_update(y)

print(x)
```

Python Dictionaries

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

Dictionaries are used to store data values in key:value pairs.

A dictionary is a collection which is ordered, changeable and do not allow duplicates.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(len(thisdict))
```

```
thisdict = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}

print(thisdict)
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(type(thisdict))
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
print(x)
```

There is also a method called `get()` that will give you the same result

```
x = thisdict.get("model")
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = thisdict.keys()

print(x)
```

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.keys()

print(x) #before the change

car["color"] = "white"

print(x) #after the change

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = thisdict.values()

print(x)
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = thisdict.items()

print(x)
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

Change Dictionary Items

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["year"] = 2018
```

Add Dictionary Items

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

Remove Dictionary Items

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.popitem()
print(thisdict)
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"]
print(thisdict)
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()
print(thisdict)
```

loop

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(thisdict[x])
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict.values():
  print(x)
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict.keys():
  print(x)
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x, y in thisdict.items():
  print(x, y)
```

Copy Dictionaries

You cannot copy a dictionary simply by typing `dict2 = dict1`, because: `dict2` will only be a *reference* to `dict1`, and changes made in `dict1` will automatically also be made in `dict2`

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = thisdict.copy()
print(mydict)
```

Nested Dictionaries

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
```

```
    "name" : "Linus",
    "year" : 2011
  }
}

print(myfamily)


child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}

myfamily = {
  "child1" : child1,
  "child2" : child2,
```

```
  "child3" : child3
}

print(myfamily)


myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}

print(myfamily["child2"]["name"])
```

Python If ... Else

- 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`

```
a = 33

b = 200


if b > a:

  print("b is greater than a")
```

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

```
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")
```

Short Hand If

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

Short Hand If ... Else

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

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

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

And

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

Or

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

Not

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

Nested If

```
x = 41

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

For Loops

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

for x in fruits:

 print(x)
```

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

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

for x in fruits:

 print(x)

 if x == "banana":

  break
```

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

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

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

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

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

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

```
#If the loop breaks, the else block is not executed.
```

Nested Loops

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

PATTERN

```
rows = 5

for i in range(rows):
    for j in range(i+1):
        print("* ", end="")
    print("\n")
```

Output

```
*
* *
* * *
* * * *
* * * * *
```

```
n = 5
for i in range(0, n):
    for j in range(n-i):
        print("*", end=" ")
    print()
```

```
* * * * *
* * * *
* * *
* *
*
```

V-Ex Tech

```
rows = 5

for i in range(rows):
    for j in range(i+1):
        print(j+1, end=" ")
    print("\n")
```

```
1

1 2

1 2 3

1 2 3 4

1 2 3 4 5
```

```
rows = 5

for i in range(rows, 0, -1):
    for j in range(0, i):
        print("* ", end=" ")

    print("\n")
```

```
* * * * *

* * * *

* * *

* *

*
```

```
for i in range(rows, 0, -1):
    for j in range(1, i+1):
        print(j, end=" ")

    print("\n")
```

```
1 2 3 4 5

1 2 3 4

1 2 3

1 2

1
```

```
rows = 5

for i in range(rows, 1, -1):
    for space in range(0, rows-i):
        print("  ", end="")
    for j in range(i, 2*i-1):
        print("* ", end="")
    for j in range(1, i-1):
        print("* ", end="")
    print()
```

```
* * * * * * * * *
  * * * * * * *
    * * * * *
      * * *
        *
```

```
a = 8
for i in range(0, 5):
    for j in range(0, a):
        print(end=" ")
    a = a - 2
    for j in range(0, i+1):
        print("* ", end="")
    print()
```

```
    *
   **
  ***
 ****
*****
```

```
for i in range(num):
    for j in range((num - i) - 1):
        print(end=" ")
    for j in range(i + 1):
        print("*", end=" ")
    print()
```

```
    *
   * *
  * * *
 * * * *
* * * * *
```

Functions

Creating a Function

```
def my_function():
  print("Hello from a function")
```

```
def my_function():
  print("Hello from a function")

my_function()
```

Arguments

Information can be passed into functions as arguments

```
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Refsnes")
```

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil")
```

Arbitrary Arguments, *args

```
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

Arbitrary Keyword Arguments, **kwargs

```
def my_function(**kid):
  print("His last name is " + kid["lname"])

my_function(fname = "Tobias", lname = "Refsnes")
```

Default Parameter Value

```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
```

```
my_function("Brazil")
```

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Lambda

A lambda function can take any number of arguments, but can only have one expression.

Syntax

`lambda` *`arguments`* `:` *`expression`*

```
x = lambda a: a + 10

print(x(5))
```

```
x = lambda a, b: a * b

print(x(5, 6))
```

```
def myfunc(n):

  return lambda a : a * n
```

```
mydoubler = myfunc(2)

print(mydoubler(11))
```

Classes and Objects

```
class MyClass:
  x = 5

print(MyClass)
```

```
class MyClass:
  x = 5

p1 = MyClass()
print(p1.x)
```

```
class Person:

  def __init__(self, name, age):

    self.name = name

    self.age = age


p1 = Person("John", 36)


print(p1.name)

print(p1.age)
```

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

p1 = Person("John", 36)

print(p1)
```

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

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

delete object

```
class Person:

 def __init__(self, name, age):

  self.name = name

  self.age = age


 def myfunc(self):

  print("Hello my name is " + self.name)


p1 = Person("John", 36)


del p1.age


print(p1.age)
```

Python Inheritance

Parent class is the class being inherited from, also called base class.

Child class is the class that inherits from another class, also called derived class.

Create a Parent Class

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

#Use the Person class to create an object, and then execute the printname method:

x = Person("John", "Doe")
x.printname()
```

Create a Child Class

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  pass

x = Student("Mike", "Olsen")
x.printname()
```

Python Iterators

```
mytuple = ("apple", "banana", "cherry")
myit = iter(mytuple)

print(next(myit))
print(next(myit))
print(next(myit))
```

Python Scope

Local Scope

```
def myfunc():
  x = 300
  print(x)

myfunc()
```

```
def myfunc():
  x = 300
  def myinnerfunc():
    print(x)
  myinnerfunc()

myfunc()
```

Global Scope

```
x = 300

def myfunc():
  print(x)

myfunc()

print(x)
```

What is a Module?

Consider a module to be the same as a code library.

Create a Module

To create a module just save the code you want in a file with the file extension `.py`:

```
def greeting(name):
  print("Hello, " + name)
```

Use a Module

Now we can use the module we just created, by using the `import` statement:

Example

Import the module named mymodule, and call the greeting function:

```
import mymodule

mymodule.greeting("Jonathan")
```

Variables in Module

The module can contain functions, as already described, but also variables of all types (arrays, dictionaries, objects etc):

Example

Save this code in the file `mymodule.py`

```
person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

```
import mymodule

a = mymodule.person1["age"]
print(a)
```

Re-naming a Module

You can create an alias when you import a module, by using the `as` keyword:

Create an alias for `mymodule` called `mx`:

```
import mymodule as mx

a = mx.person1["age"]
print(a)
```

Built-in Modules

There are several built-in modules in Python, which you can import whenever you like.

```
import platform
```

```
x = platform.system()
print(x)
```

Using the dir() Function

```
import platform
```

```
x = dir(platform)
print(x)
```

```
def greeting(name):
  print("Hello, " + name)

person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

Import only the person1 dictionary from the module:

```
from mymodule import person1

print (person1["age"])
```

Datetime

```
import datetime

x = datetime.datetime.now()

print(x)
```

```
import datetime

x = datetime.datetime.now()

print(x.year)

print(x.strftime("%A"))
```

https://www.w3schools.com/python/python_datetime.asp

Python Math

```
x = min(5, 10, 25)
y = max(5, 10, 25)

print(x)
print(y)
```

```
x = pow(4, 3)

print(x)
```

```
import math

x = math.sqrt(64)

print(x)
```

```
import math

x = math.pi

print(x)
```

JSON in Python

Convert from JSON to Python

```
import json

# some JSON:
x = '{ "name":"John", "age":30, "city":"New York"}'

# parse x:
y = json.loads(x)

# the result is a Python dictionary:
print(y["age"])
```

Convert from Python to JSON

If you have a Python object, you can convert it into a JSON string by using the `json.dumps()` method.

```
import json

# a Python object (dict):
x = {
  "name": "John",
  "age": 30,
  "city": "New York"
}

# convert into JSON:
y = json.dumps(x)

# the result is a JSON string:
print(y)
```

RegEx

A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern.

```
import re

#Check if the string starts with "The" and ends with "Spain":

txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)

if x:
  print("YES! We have a match!")
else:
  print("No match")
```

Character	Description	Example
[]	A set of characters	"[a-m]"

V-Ex Tech

\	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"
^	Starts with	"^hello"
$	Ends with	"planet$"
*	Zero or more occurrences	"he.*o"
+	One or more occurrences	"he.+o"
?	Zero or one occurrences	"he.?o"
{}	Exactly the specified number of occurrences	"he.{2}o"
|	Either or	"falls|stays"

()	Capture and group

The findall() Function

The `findall()` function returns a list containing all matches

```
import re

txt = "The rain in Spain"
x = re.findall("ai", txt)
print(x)
```

The list contains the matches in the order they are found.

If no matches are found, an empty list is returned:

```
import re

txt = "The rain in Spain"

#Check if "Portugal" is in the string:

x = re.findall("Portugal", txt)
```

```
print(x)
```

```
if (x):
  print("Yes, there is at least one match!")
else:
  print("No match")
```

The search() Function

The `search()` function searches the string for a match, and returns a Match object if there is a match.

If there is more than one match, only the first occurrence of the match will be returned:

```
import re

txt = "The rain in Spain"
x = re.search("\s", txt)

print("The first white-space character is located in position:",
x.start())
```

If no matches are found, the value `None` is returned:

```
import re

txt = "The rain in Spain"
x = re.search("Portugal", txt)
print(x)
```

The split() Function

```
import re

#Split the string at every white-space character:

txt = "The rain in Spain"
x = re.split("\s", txt)
print(x)
```

```
import re

#Split the string at the first white-space character:

txt = "The rain in Spain"
x = re.split("\s", txt, 1)
print(x)
```

The sub() Function

The `sub()` function replaces the matches with the text of your choice:

```
import re

#Replace all white-space characters with the digit "9":

txt = "The rain in Spain"
x = re.sub("\s", "9", txt)
print(x)
```

```
import re

#Replace the first two occurrences of a white-space character with the digit 9:

txt = "The rain in Spain"
x = re.sub("\s", "9", txt, 2)
print(x)
```

`.span()` returns a tuple containing the start-, and end positions of the match.

`.group()` returns the part of the string where there was a match

```
import re

#Search for an upper case "S" character in the beginning of a word, and print its position:

txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.span())
```

```
import re

#Search for an upper case "S" character in the beginning of a word, and print the word:

txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.group())
```

What is PIP?

PIP is a package manager for Python packages, or modules if you like

What is a Package?

A package contains all the files you need for a module.

Modules are Python code libraries you can include in your project.

Check if PIP is Installed

```
pip –version
```

```
import camelcase

c = camelcase.CamelCase()

txt = "lorem ipsum dolor sit amet"

print(c.hump(txt))

#This method capitalizes the first letter of each word.
```

V-Ex Tech

Write your name

```
import turtle

turtle.color('purple')
style = ('Courier', 90, 'normal')
turtle.write('PRATIBHA', font=style, align='center')
turtle.hideturtle()
t = turtle.Turtle()
t.reset()
t.pencolor('purple')
t.pensize(5)
t.penup()
t.goto(-300,200)
# p
t.pendown()
t.fd(20)
t.circle(-30, 180)
t.fd(20)
t.rt(90)
t.fd(60)
t.bk(60)
t.lt(180)
t.fd(60)

t.penup()
t.goto(-230, 200)

# R
t.pendown()
t.lt(90)
t.fd(20)
t.circle(-30, 180)
t.fd(20)
t.rt(90)
t.fd(60)
t.bk(60)
t.lt(180)
t.fd(60)
t.bk(60)
t.lt(45)
t.fd(80)
t.rt(45)

t.penup()
t.goto(-160, 200)
```

Try Except

#The try block will generate an error, because x is not defined:

```
try:
  print(x)
except:
  print("An exception occurred")
```

#The try block does not raise any errors, so the else block is executed:

```
try:
  print("Hello")
except:
  print("Something went wrong")
else:
  print("Nothing went wrong")
```

V-Ex Tech

User Input

```
username = input("Enter username:")

print("Username is: " + username)
```

add two number

```
# Python3 program to add two numbers

number1 = input("First number: ")
number2 = input("\nSecond number: ")

# Adding two numbers
# User might also enter float numbers
sum = int(number1) + int(number2)

# Display the sum
# will print value in float
print("The sum of {0} and {1} is {2}" .format(number1,number2, sum))
```

File Handling

File Open

File handling is an important part of any web application.

Python has several functions for creating, reading, updating, and deleting files.

There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

```
f = open("demofile.txt")
```

demofile.txt

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

```
f = open("demofile.txt", "r")

print(f.read())
```

open file in different location

```
f = open("D:\\myfiles\welcome.txt", "r")
print(f.read())
```

```
f = open("demofile.txt", "r")
print(f.read(5))
```

read one line

```
f = open("demofile.txt", "r")

print(f.readline())
print(f.readline())
```

or else

```
f = open("demofile.txt", "r")
for x in f:
  print(x)
```

close the file

```
f = open("demofile.txt", "r")

print(f.readline())

f.close()
```

File Write

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```

```
f = open("demofile3.txt", "w")

f.write("Woops! I have deleted the content!")

f.close()


#open and read the file after the overwriting:

f = open("demofile3.txt", "r")

print(f.read())
```

Create a New File

`x"` - Create - will create a file, returns an error if the file exist

`"a"` - Append - will create a file if the specified file does not exist

`"w"` - Write - will create a file if the specified file does not exist

```
f = open("myfile.txt", "x")
```

```
f = open("myfile.txt", "w")
```

Delete File

To delete a file, you must import the OS module, and run its `os.remove()` function:

```
import os
os.remove("demofile.txt")
```

```
import os
if os.path.exists("demofile.txt"):
  os.remove("demofile.txt")
else:
  print("The file does not exist")
```

#The finally block gets executed no matter if the try block raises any errors or not:

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```