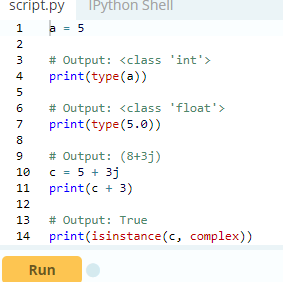
# **CHAPTER 5**

# **PYTHON DATA TYPES**

**Number Data Type in Python**

Python supports integers, floating point numbers and complex numbers. They are defined as int, float and complex class in Python. Integers and floating points are separated by the presence or absence of a decimal point. 5 is integer whereas 5.0 is a floating point number.

Complex numbers are written in the form, x + yj, where x is the real part and y is the imaginary part. We can use the type() function to know which class a variable or a value belongs to and isinstance() function to check if it belongs to a particular class.

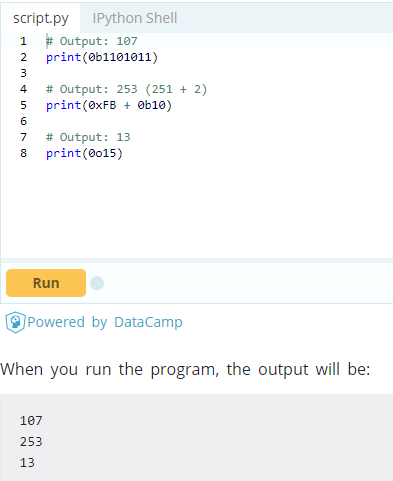


While integers can be of any length, a floating point number is accurate only up to 15 decimal places (the 16th place is inaccurate). Numbers we deal with everyday are decimal (base 10) number system. But computer programmers (generally embedded programmer) need to work with binary (base 2), hexadecimal (base 16) and octal (base 8) number systems.

In Python, we can represent these numbers by appropriately placing a prefix before that number. Following table lists these prefix.

|  |  |
| --- | --- |
| Number System | Prefix |
| Binary | '0b' or '0B' |
| Octal | '0o' or '0O' |
| Hexadecimal | '0x' or '0X' |
| Number system prefix for Python numbers | |

Here are some examples



**Type Conversion**

We can convert one type of number into another. This is also known as coercion. Operations like addition, subtraction coerce integer to float implicitly (automatically), if one of the operand is float.

1. >>> 1 + 2.0
2. 3.0

We can see above that 1 (integer) is coerced into 1.0 (float) for addition and the result is also a floating point number. We can also use built-in functions like int(), float() and complex() to convert between types explicitly. These function can even convert from [strings](https://www.programiz.com/python-programming/string).

1. >>> int(2.3)
2. 2
3. >>> int(-2.8)
4. -2
5. >>> float(5)
6. 5.0
7. >>> complex('3+5j')
8. (3+5j)

When converting from float to integer, the number gets truncated (integer that is closer to zero).

**Python Decimal**

Python built-in class float performs some calculations that might amaze us. We all know that the sum of 1.1 and 2.2 is 3.3, but Python seems to disagree.

1. >>> (1.1 + 2.2) == 3.3
2. False

**What is going on?**

It turns out that floating-point numbers are implemented in computer hardware as binary fractions, as computer only understands binary (0 and 1). Due to this reason, most of the decimal fractions we know, cannot be accurately stored in our computer.

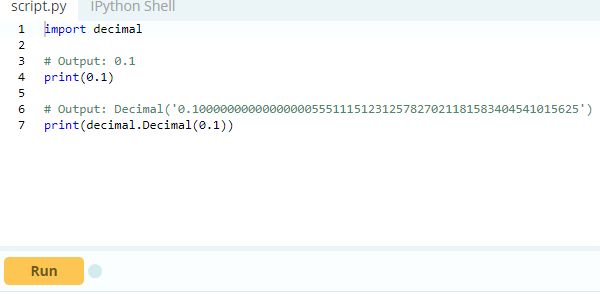
Let's take an example. We cannot represent the fraction 1/3 as a decimal number. This will give 0.33333333... which is infinitely long, and we can only approximate it.

Turns out decimal fraction 0.1 will result into an infinitely long binary fraction of 0.000110011001100110011... and our computer only stores a finite number of it.

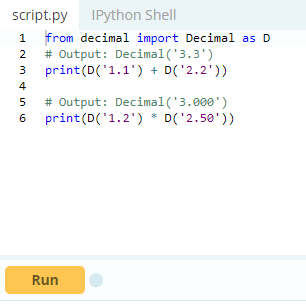
This will only approximate 0.1 but never be equal. Hence, it is the limitation of our computer hardware and not an error in Python.

1. >>> 1.1 + 2.2
2. 3.3000000000000003

To overcome this issue, we can use decimal module that comes with Python. While floating point numbers have precision up to 15 decimal places, the decimal module has user settable precision.



This module is used when we want to carry out decimal calculations like we learned in school. It also preserves significance. We know 25.50 kg is more accurate than 25.5 kg as it has two significant decimal places compared to one.



Notice the trailing zeroes in the above example. We might ask, why not implement Decimal every time, instead of float? The main reason is efficiency. Floating point operations are carried out must faster than Decimal operations.

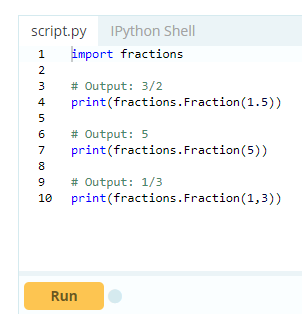
**When to use Decimal instead of float?**

We generally use Decimal in the following cases.

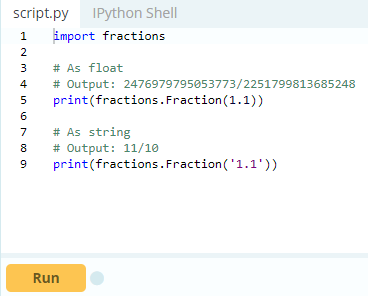
* When we are making financial applications that need exact decimal representation.
* When we want to control the level of precision required.
* When we want to implement the notion of significant decimal places.
* When we want the operations to be carried out like we did at school

**Python Fractions**

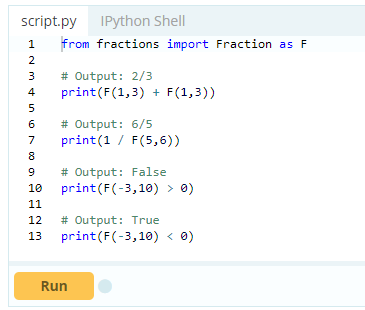
Python provides operations involving fractional numbers through its fractions module.A fraction has a numerator and a denominator, both of which are integers. This module has support for rational number arithmetic. We can create Fraction objects in various ways.



While creating Fraction from float, we might get some unusual results. This is due to the imperfect binary floating point number representation as discussed in the previous section. Fortunately, Fraction allows us to instantiate with string as well. This is the preferred options when using decimal numbers.

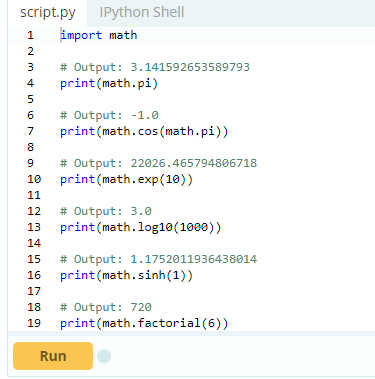


This datatype supports all basic operations. Here are few examples.

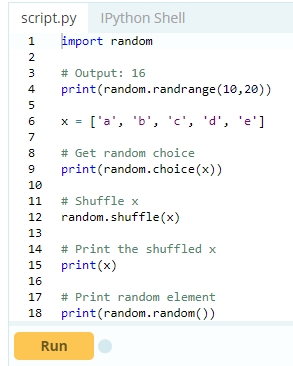


**Python Mathematics**

Python offers modules like math and random to carry out different mathematics like trigonometry, logarithms, probability and statistics, etc.



Here is the full list functions and attributes available in [Python math module](https://www.programiz.com/python-programming/modules/math).



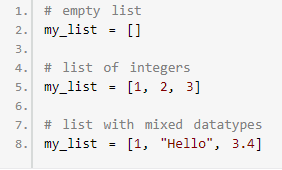
Here is the full list functions and attributes available in [Python random module](https://www.programiz.com/python-programming/modules/random).

# **Python List**

# Python offers a range of compound datatypes often referred to as sequences. List is one of the most frequently used and very versatile datatype used in Python.

## How to create a list?

In Python programming, a list is created by placing all the items (elements) inside a square bracket [ ], separated by commas. It can have any number of items and they may be of different types (integer, float, string etc.).



Also, a list can even have another list as an item. This is called nested list.

# nested list

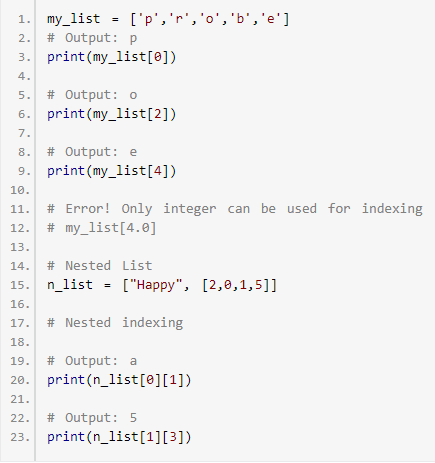
my\_list = ["mouse", [8, 4, 6], ['a']]

## How to access elements from a list?

There are various ways in which we can access the elements of a list.

### List Index

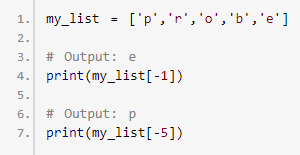
We can use the index operator [] to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4. Trying to access an element other that this will raise an IndexError. The index must be an integer. We can't use float or other types, this will result into TypeError. Nested list are accessed using nested indexing.



### 

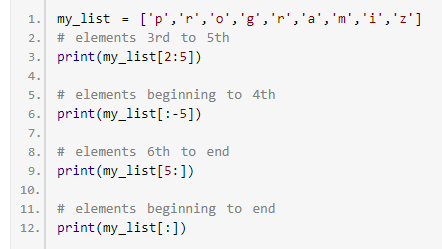
### Negative indexing

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.



## How to slice lists in Python?

We can access a range of items in a list by using the slicing operator (colon).



Slicing can be best visualized by considering the index to be between the elements as shown below. So if we want to access a range, we need two index that will slice that portion from the list.



## How to change or add elements to a list?

List are mutable, meaning, their elements can be changed unlike [string](https://www.programiz.com/python-programming/string) or [tuple](https://www.programiz.com/python-programming/tuple). We can use assignment operator (=) to change an item or a range of items.

1. # mistake values
2. odd = [2, 4, 6, 8]
3. # change the 1st item
4. odd[0] = 1
5. # Output: [1, 4, 6, 8]
6. print(odd)
7. # change 2nd to 4th items
8. odd[1:4] = [3, 5, 7]
9. # Output: [1, 3, 5, 7]
10. print(odd)

We can add one item to a list using append() method or add several items using extend()method.

1. odd = [1, 3, 5]
2. odd.append(7)
3. # Output: [1, 3, 5, 7]
4. print(odd)
5. odd.extend([9, 11, 13])
6. # Output: [1, 3, 5, 7, 9, 11, 13]
7. print(odd)

We can also use + operator to combine two lists. This is also called concatenation. The \* operator repeats a list for the given number of times.

1. odd = [1, 3, 5]
2. # Output: [1, 3, 5, 9, 7, 5]
3. print(odd + [9, 7, 5])
4. #Output: ["re", "re", "re"]
5. print(["re"] \* 3)

Furthermore, we can insert one item at a desired location by using the method insert() or insert multiple items by squeezing it into an empty slice of a list.

1. odd = [1, 9]
2. odd.insert(1,3)
3. # Output: [1, 3, 9]
4. print(odd)
5. odd[2:2] = [5, 7]
6. # Output: [1, 3, 5, 7, 9]
7. print(odd)

## How to delete or remove elements from a list?

We can delete one or more items from a list using the keyword del. It can even delete the list entirely.

1. my\_list = ['p','r','o','b','l','e','m']
2. # delete one item
3. del my\_list[2]
4. # Output: ['p', 'r', 'b', 'l', 'e', 'm']
5. print(my\_list)
6. # delete multiple items
7. del my\_list[1:5]
8. # Output: ['p', 'm']
9. print(my\_list)
10. # delete entire list
11. del my\_list
12. # Error: List not defined
13. print(my\_list)

We can use remove() method to remove the given item or pop() method to remove an item at the given index. The pop() method removes and returns the last item if index is not provided. This helps us implement lists as stacks (first in, last out data structure).

We can also use the clear() method to empty a list.

1. my\_list = ['p','r','o','b','l','e','m']
2. my\_list.remove('p')
3. # Output: ['r', 'o', 'b', 'l', 'e', 'm']
4. print(my\_list)
5. # Output: 'o'
6. print(my\_list.pop(1))
7. # Output: ['r', 'b', 'l', 'e', 'm']
8. print(my\_list)
9. # Output: 'm'
10. print(my\_list.pop())
11. # Output: ['r', 'b', 'l', 'e']
12. print(my\_list)
13. my\_list.clear()
14. # Output: []
15. print(my\_list)

Finally, we can also delete items in a list by assigning an empty list to a slice of elements.

1. >>> my\_list = ['p','r','o','b','l','e','m']
2. >>> my\_list[2:3] = []
3. >>> my\_list
4. ['p', 'r', 'b', 'l', 'e', 'm']
5. >>> my\_list[2:5] = []
6. >>> my\_list
7. ['p', 'r', 'm']

## Python List Methods

Methods that are available with list object in Python programming are tabulated below. They are accessed as list.method(). Some of the methods have already been used above.

|  |
| --- |
| [Python List Methods](https://www.programiz.com/python-programming/methods/list) |
| [**append()** - Add an element to the end of the list](https://www.programiz.com/python-programming/methods/list/append) |
| [**extend()** - Add all elements of a list to the another list](https://www.programiz.com/python-programming/methods/list/extend) |
| [**insert()** - Insert an item at the defined index](https://www.programiz.com/python-programming/methods/list/insert) |
| [**remove()** - Removes an item from the list](https://www.programiz.com/python-programming/methods/list/remove) |
| [**pop()** - Removes and returns an element at the given index](https://www.programiz.com/python-programming/methods/list/pop) |
| [**clear()** - Removes all items from the list](https://www.programiz.com/python-programming/methods/list/clear) |
| [**index()** - Returns the index of the first matched item](https://www.programiz.com/python-programming/methods/list/index) |
| [**count()** - Returns the count of number of items passed as an argument](https://www.programiz.com/python-programming/methods/list/count) |
| [**sort()** - Sort items in a list in ascending order](https://www.programiz.com/python-programming/methods/list/sort) |
| [**reverse()** - Reverse the order of items in the list](https://www.programiz.com/python-programming/methods/list/reverse) |
| [**copy()** - Returns a shallow copy of the list](https://www.programiz.com/python-programming/methods/list/copy) |

Some examples of Python list methods:

1. my\_list = [3, 8, 1, 6, 0, 8, 4]
2. # Output: 1
3. print(my\_list.index(8))
4. # Output: 2
5. print(my\_list.count(8))
6. my\_list.sort()
7. # Output: [0, 1, 3, 4, 6, 8, 8]
8. print(my\_list)
9. my\_list.reverse()
10. # Output: [8, 8, 6, 4, 3, 1, 0]
11. print(my\_list)

## List Comprehension: Elegant way to create new List

List comprehension is an elegant and concise way to create new list from an existing list in Python. List comprehension consists of an expression followed by [for statement](https://www.programiz.com/python-programming/for-loop) inside square brackets. Here is an example to make a list with each item being increasing power of 2.

1. pow2 = [2 \*\* x for x in range(10)]
2. # Output: [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
3. print(pow2)

This code is equivalent to

1. pow2 = []
2. for x in range(10):
3. pow2.append(2 \*\* x)

A list comprehension can optionally contain more for or [if statements](https://www.programiz.com/python-programming/if-elif-else). An optional ifstatement can filter out items for the new list. Here are some examples.

1. >>> pow2 = [2 \*\* x for x in range(10) if x > 5]
2. >>> pow2
3. [64, 128, 256, 512]
4. >>> odd = [x for x in range(20) if x % 2 == 1]
5. >>> odd
6. [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
7. >>> [x+y for x in ['Python ','C '] for y in ['Language','Programming']]
8. ['Python Language', 'Python Programming', 'C Language', 'C Programming']

## Other List Operations in Python

### List Membership Test

We can test if an item exists in a list or not, using the keyword in.

1. my\_list = ['p','r','o','b','l','e','m']
2. # Output: True
3. print('p' in my\_list)
4. # Output: False
5. print('a' in my\_list)
6. # Output: True
7. print('c' not in my\_list)

### Iterating Through a List

Using a for loop we can iterate though each item in a list.

1. for fruit in ['apple','banana','mango']:
2. print("I like",fruit)

# **Python Tuple**

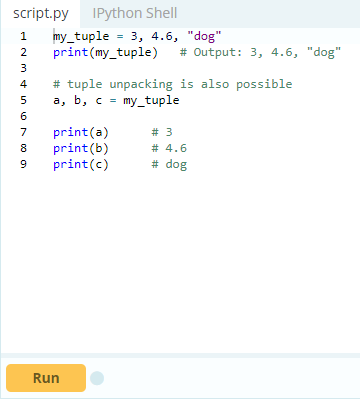
A tuple in Python is similar to a [list](https://www.programiz.com/python-programming/list). The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas, in a list, elements can be changed.

## Creating a Tuple

A tuple is created by placing all the items (elements) inside parentheses (), separated by commas. The parentheses are optional, however, it is a good practice to use them. A tuple can have any number of items and they may be of different types (integer, float, list, [string](https://www.programiz.com/python-programming/string), etc.).

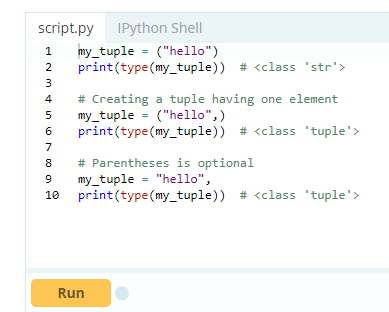


A tuple can also be created without using parentheses. This is known as tuple packing.



Creating a tuple with one element is a bit tricky.

Having one element within parentheses is not enough. We will need a trailing comma to indicate that it is, in fact, a tuple.



## Access Tuple Elements

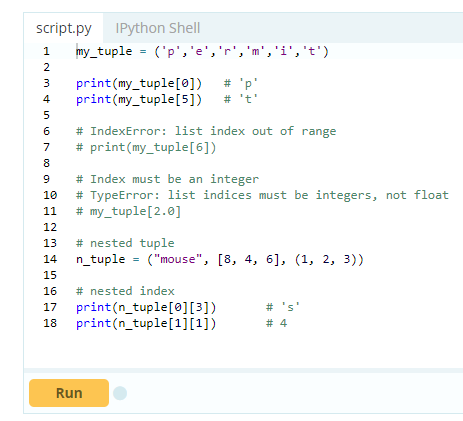
There are various ways in which we can access the elements of a tuple.

### Indexing

We can use the index operator [] to access an item in a tuple where the index starts from 0.

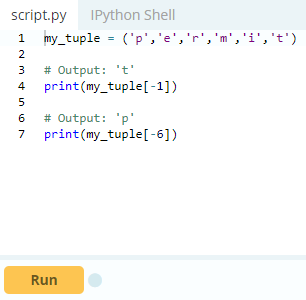
So, a tuple having 6 elements will have indices from 0 to 5. Trying to access an element outside of tuple (for example, 6, 7,...) will raise an IndexError.

The index must be an integer; so we cannot use float or other types. This will result in TypeError. Likewise, nested tuples are accessed using nested indexing, as shown in the example below.



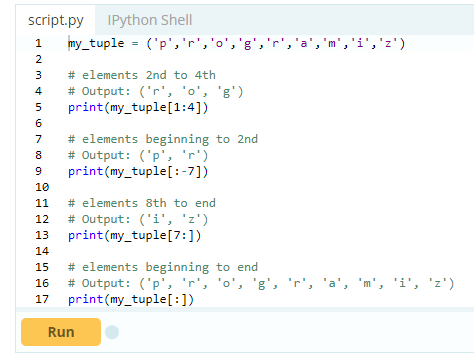
### 2. Negative Indexing

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.



### 3. Slicing

We can access a range of items in a tuple by using the slicing operator - colon ":".

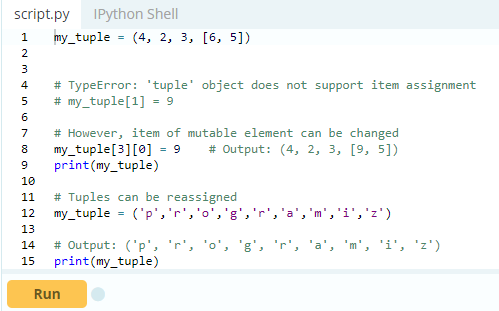


Slicing can be best visualized by considering the index to be between the elements as shown below. So if we want to access a range, we need the index that will slice the portion from the tuple.



## Changing a Tuple

Unlike lists, tuples are immutable. This means that elements of a tuple cannot be changed once it has been assigned. But, if the element is itself a mutable datatype like list, its nested items can be changed. We can also assign a tuple to different values (reassignment).

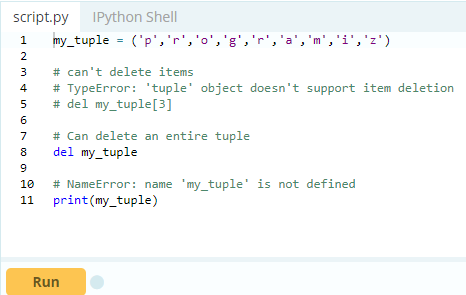


We can use + operator to combine two tuples. This is also called **concatenation**. We can also **repeat**the elements in a tuple for a given number of times using the \*operator. Both + and \* operations result in a new tuple.



## Deleting a Tuple

As discussed above, we cannot change the elements in a tuple. That also means we cannot delete or remove items from a tuple. But deleting a tuple entirely is possible using the keyword [del](https://www.programiz.com/python-programming/keyword-list#del).

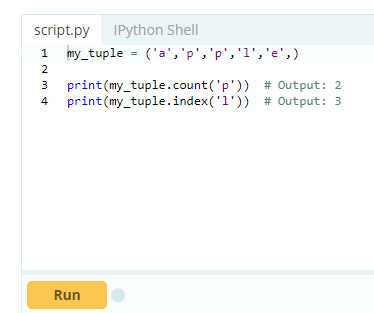


## Tuple Methods

Methods that add items or remove items are not available with tuple. Only the following two methods are available.

|  |  |
| --- | --- |
| Method | Description |
| [count(x)](https://www.programiz.com/python-programming/methods/tuple/count) | Returns the number of items x |
| [index(x)](https://www.programiz.com/python-programming/methods/tuple/index) | Returns the index of the first item that is equal to x |
| Python Tuple Method | |

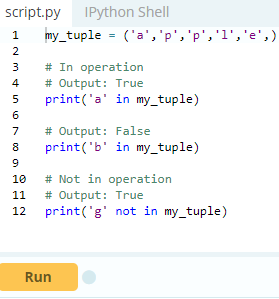
Some examples of Python tuple methods:



## Other Tuple Operations

### 1. Tuple Membership Test

We can test if an item exists in a tuple or not, using the keyword in.



### Iterating Through a Tuple

Using a for loop we can iterate through each item in a tuple.



Advantages of Tuple over List

Since tuples are quite similar to lists, both of them are used in similar situations as well. However, there are certain advantages of implementing a tuple over a list. Below listed are some of the main advantages:

* We generally use tuple for heterogeneous (different) datatypes and list for homogeneous (similar) datatypes.
* Since tuples are immutable, iterating through tuple is faster than with list. So there is a slight performance boost.
* Tuples that contain immutable elements can be used as a key for a dictionary. With lists, this is not possible.
* If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

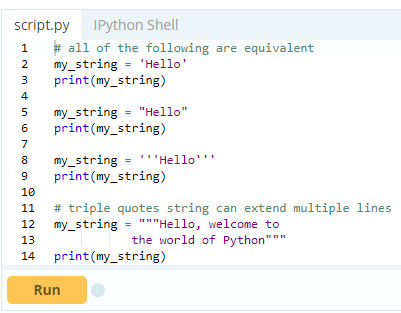
# **Python Strings**

A string is a sequence of characters. A character is simply a symbol. For example, the English language has 26 characters. Computers do not deal with characters, they deal with numbers (binary). Even though you may see characters on your screen, internally it is stored and manipulated as a combination of 0's and 1's.

This conversion of character to a number is called encoding, and the reverse process is decoding. ASCII and Unicode are some of the popular encoding used. In Python, string is a sequence of Unicode character. Unicode was introduced to include every character in all languages and bring uniformity in encoding. You can [learn more about Unicode](http://docs.python.org/3.3/howto/unicode.html) from here.

## 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.



When you run the program, the output will be:

Hello

Hello

Hello

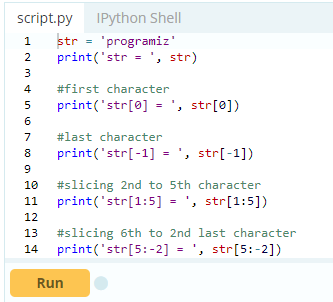
Hello, welcome to

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. 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 float 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).



If we try to access index out of the range or use decimal number, we will get errors.

1. # index must be in range
2. >>> my\_string[15]
3. ...
4. IndexError: string index out of range
5. # index must be an integer
6. >>> my\_string[1.5]
7. ...
8. TypeError: string indices must be integers

Slicing can be best visualized by considering the index to be between the elements as shown below. If we want to access a range, we need the index that will slice the portion from the string.



## How to change or delete a string?

Strings are immutable. This means that elements of a string cannot be changed once it has been assigned. We can simply reassign different strings to the same name.

1. >>> my\_string = 'programiz'
2. >>> my\_string[5] = 'a'
3. ...
4. TypeError: 'str' object does not support item assignment
5. >>> my\_string = 'Python'
6. >>> my\_string
7. 'Python'

We cannot delete or remove characters from a string. But deleting the string entirely is possible using the keyword del.

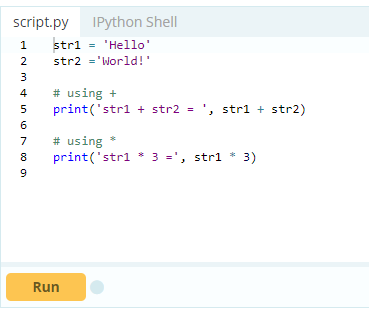
1. >>> del my\_string[1]
2. ...
3. TypeError: 'str' object doesn't support item deletion
4. >>> del my\_string
5. >>> my\_string
6. ...
7. NameError: name 'my\_string' is not defined

## Python String Operations

There are many operations that can be performed with string which makes it one of the most used [datatypes in Python](https://www.programiz.com/python-programming/variables-datatypes).

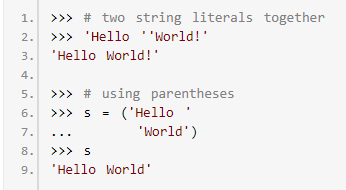
### Concatenation of Two or More Strings

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 the string for a given number of times.



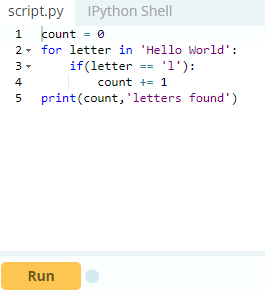
Writing two string literals together also concatenates them like **+** operator.

If we want to concatenate strings in different lines, we can use parentheses.



### Iterating Through String

Using [for loop](https://www.programiz.com/python-programming/for-loop) we can iterate through a string. Here is an example to count the number of 'l' in a string.



### String Membership Test

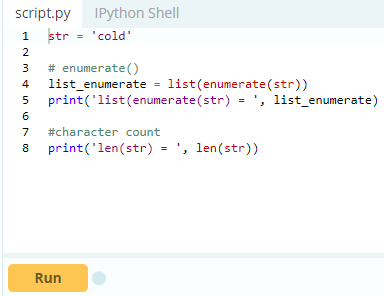
We can test if a sub string exists within a string or not, using the keyword in.

1. >>> 'a' in 'program'
2. True
3. >>> 'at' not in 'battle'
4. False

### 

### Built-in functions to Work with Python

Various built-in functions that work with sequence, works with string as well. Some of the commonly used ones are enumerate() and len(). The enumerate() function returns an enumerate object. It contains the index and value of all the items in the string as pairs. This can be useful for iteration. Similarly, len() returns the length (number of characters) of the string.



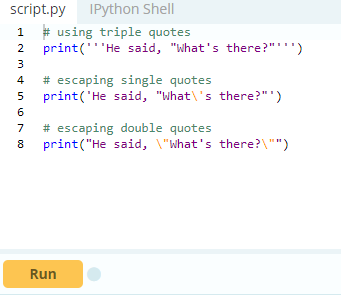
## Python String Formatting

### Escape Sequence

If we want to print a text like -He said, "What's there?"- we can neither use single quote or double quotes. This will result into SyntaxError as the text itself contains both single and double quotes.

1. >>> print("He said, "What's there?"")
2. ...
3. SyntaxError: invalid syntax
4. >>> print('He said, "What's there?"')
5. ...
6. SyntaxError: invalid syntax

One way to get around this problem is to use triple quotes. Alternatively, we can use escape sequences. An escape sequence starts with a backslash and is interpreted differently. If we use single quote to represent a string, all the single quotes inside the string must be escaped. Similar is the case with double quotes. Here is how it can be done to represent the above text.



Here is a list of all the escape sequence supported by Python.

|  |  |
| --- | --- |
| Escape Sequence | Description |
| \newline | Backslash and newline ignored |
| \\ | Backslash |
| \' | Single quote |
| \" | Double quote |
| \a | ASCII Bell |
| \b | ASCII Backspace |
| \f | ASCII Formfeed |
| \n | ASCII Linefeed |
| \r | ASCII Carriage Return |
| \t | ASCII Horizontal Tab |
| \v | ASCII Vertical Tab |
| \ooo | Character with octal value ooo |
| \xHH | Character with hexadecimal value HH |
| **Escape Sequence in Python** | |

Here are some examples

1. >>> print("C:\\Python32\\Lib")
2. C:\Python32\Lib
3. >>> print("This is printed\nin two lines")
4. This is printed
5. in two lines
6. >>> print("This is \x48\x45\x58 representation")
7. This is HEX representation

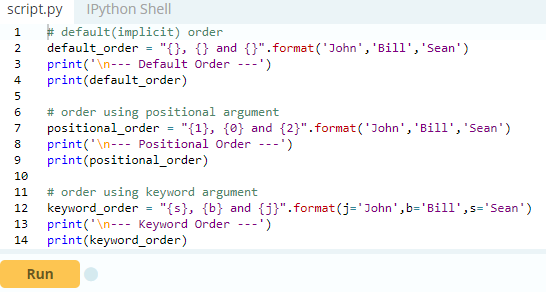
### Raw String to ignore escape sequence

Sometimes we may wish to ignore the escape sequences inside a string. To do this we can place r or R in front of the string. This will imply that it is a raw string and any escape sequence inside it will be ignored.

1. >>> print("This is \x61 \ngood example")
2. This is a
3. good example
4. >>> print(r"This is \x61 \ngood example")
5. This is \x61 \ngood example

### The format() Method for Formatting Strings

The format() method that is available with the string object is very versatile and powerful in formatting strings. Format strings contains curly braces {} as placeholders or replacement fields which gets replaced. We can use positional arguments or keyword arguments to specify the order.



The format() method can have optional format specifications. They are separated from field name using colon. For example, we can left-justify <, right-justify > or center ^ a string in the given space. We can also format integers as binary, hexadecimal etc. and floats can be rounded or displayed in the exponent format. There are a ton of formatting you can use. Visit here for all the [string formatting available with the format()](https://www.programiz.com/python-programming/methods/string/format) method.

1. >>> # formatting integers
2. >>> "Binary representation of {0} is {0:b}".format(12)
3. 'Binary representation of 12 is 1100'
4. >>> # formatting floats
5. >>> "Exponent representation: {0:e}".format(1566.345)
6. 'Exponent representation: 1.566345e+03'
7. >>> # round off
8. >>> "One third is: {0:.3f}".format(1/3)
9. 'One third is: 0.333'
10. >>> # string alignment
11. >>> "|{:<10}|{:^10}|{:>10}|".format('butter','bread','ham')
12. '|butter | bread | ham|'

### 

### Old style formatting

We can even format strings like the old sprintf() style used in C programming language. We use the % operator to accomplish this.

1. >>> x = 12.3456789
2. >>> print('The value of x is %3.2f' %x)
3. The value of x is 12.35
4. >>> print('The value of x is %3.4f' %x)
5. The value of x is 12.3457

## Common Python String Methods

There are numerous methods available with the string object. The format() method that we mentioned above is one of them. Some of the commonly used methods are lower(), upper(), join(), split(), find(), replace() etc. Here is a complete list of all the [built-in methods to work with strings in Python](https://www.programiz.com/python-programming/methods/string).

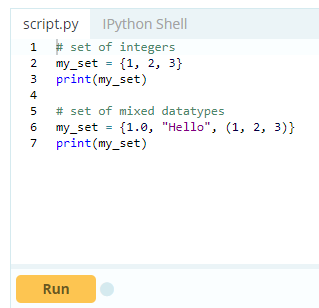
1. >>> "PrOgRaMiZ".lower()
2. 'programiz'
3. >>> "PrOgRaMiZ".upper()
4. 'PROGRAMIZ'
5. >>> "This will split all words into a list".split()
6. ['This', 'will', 'split', 'all', 'words', 'into', 'a', 'list']
7. >>> ' '.join(['This', 'will', 'join', 'all', 'words', 'into', 'a', 'string'])
8. 'This will join all words into a string'
9. >>> 'Happy New Year'.find('ew')
10. 7
11. >>> 'Happy New Year'.replace('Happy','Brilliant')
12. 'Brilliant New Year'

# **Python Sets**

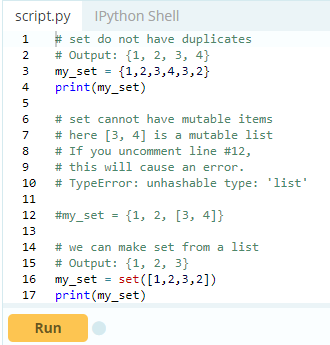
A set is an unordered collection of items. Every element is unique (no duplicates) and must be immutable (which cannot be changed). However, the set itself is mutable. We can add or remove items from it. Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc.

## How to create a set?

A set is created by placing all the items (elements) inside curly braces {}, separated by comma or by using the built-in function set(). It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have a mutable element, like [list](https://www.programiz.com/python-programming/list), set or [dictionary](https://www.programiz.com/python-programming/dictionary), as its element.

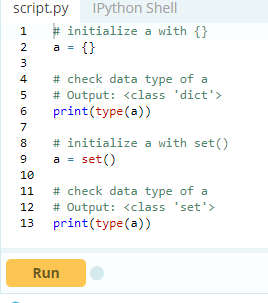


Try the following examples as well.



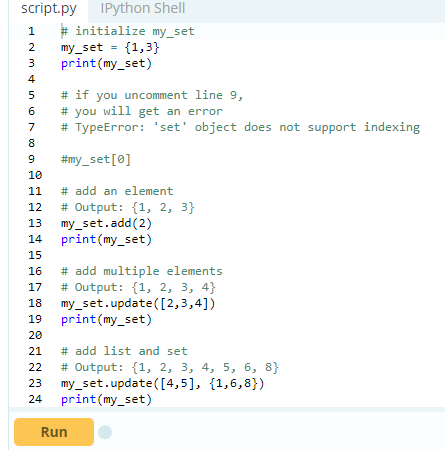
Creating an empty set is a bit tricky.

Empty curly braces {} will make an empty dictionary in Python. To make a set without any elements we use the set() function without any argument.



## How to change a set in Python?

Sets are mutable. But since they are unordered, indexing have no meaning. We cannot access or change an element of set using indexing or slicing. Set does not support it. We can add single element using the add() method and multiple elements using the update() method. The update() method can take [tuples](https://www.programiz.com/python-programming/tuple), lists, [strings](https://www.programiz.com/python-programming/string) or other sets as its argument. In all cases, duplicates are avoided.



When you run the program, the output will be:

{1, 3}

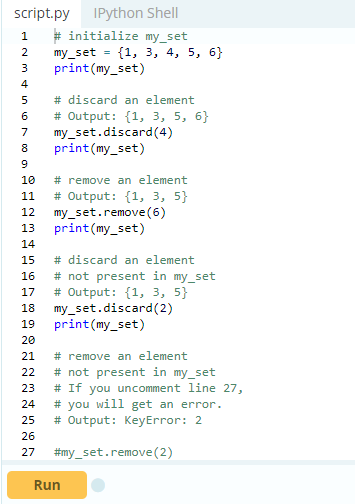
{1, 2, 3}

{1, 2, 3, 4}

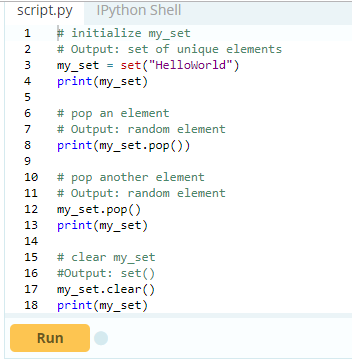
{1, 2, 3, 4, 5, 6, 8}

## How to remove elements from a set?

A particular item can be removed from set using methods, discard() and remove(). The only difference between the two is that, while using discard() if the item does not exist in the set, it remains unchanged. But remove() will raise an error in such condition. The following example will illustrate this.



Similarly, we can remove and return an item using the pop() method. Set being unordered, there is no way of determining which item will be popped. It is completely arbitrary. We can also remove all items from a set using clear().



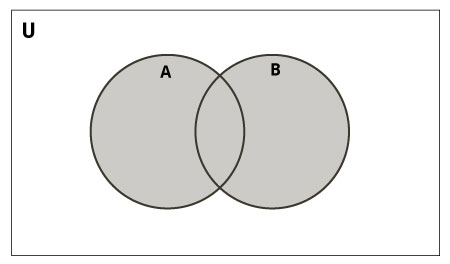
## Python Set Operations

Sets can be used to carry out mathematical set operations like union, intersection, difference and symmetric difference. We can do this with operators or methods. Let us consider the following two sets for the following operations.

1. >>> A = {1, 2, 3, 4, 5}
2. >>> B = {4, 5, 6, 7, 8}

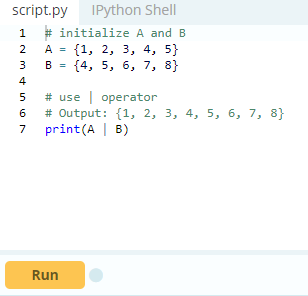
### 

### Set Union



Union of A and B is a set of all elements from both sets.

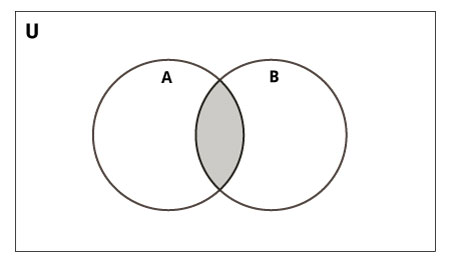
Union is performed using | operator. Same can be accomplished using the method union().



Try the following examples on Python shell.

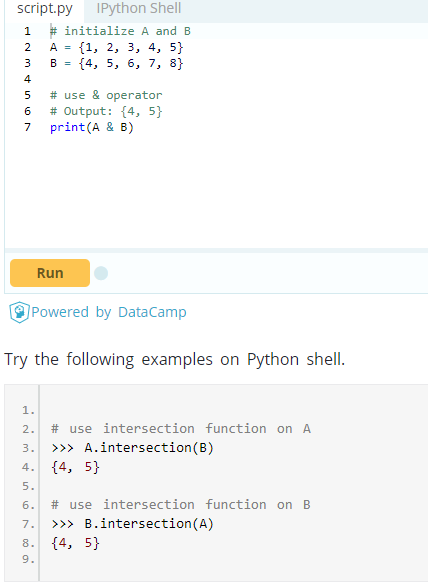
1. # use union function
2. >>> A.union(B)
3. {1, 2, 3, 4, 5, 6, 7, 8}
4. # use union function on B
5. >>> B.union(A)
6. {1, 2, 3, 4, 5, 6, 7, 8}

### Set Intersection

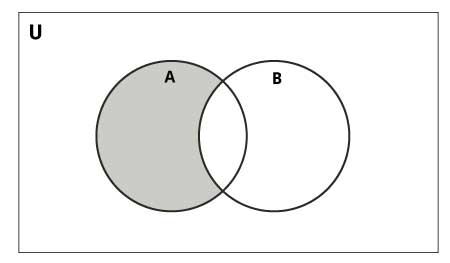


Intersection of A and B is a set of elements that are common in both sets.

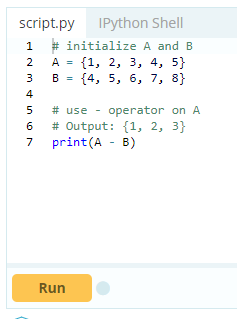
Intersection is performed using & operator. Same can be accomplished using the method intersection().



### Set Difference



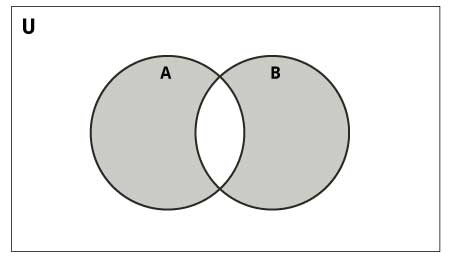
Difference of A and B (A - B) is a set of elements that are only in A but not in B. Similarly, B - A is a set of element in B but not in A. Difference is performed using - operator. Same can be accomplished using the method difference().



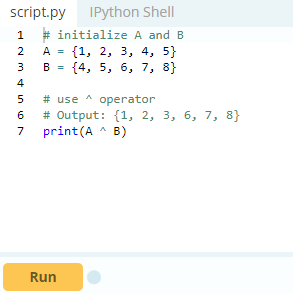
Try the following examples on Python shell.

1. # use difference function on A
2. >>> A.difference(B)
3. {1, 2, 3}
4. # use - operator on B
5. >>> B - A
6. {8, 6, 7}
7. # use difference function on B
8. >>> B.difference(A)
9. {8, 6, 7}

Set Symmetric Difference



Symmetric Difference of A and B is a set of elements in both A and B except those that are common in both. Symmetric difference is performed using ^ operator. Same can be accomplished using the method symmetric\_difference().



Try the following examples on Python shell.

1. # use symmetric\_difference function on A
2. >>> A.symmetric\_difference(B)
3. {1, 2, 3, 6, 7, 8}
4. # use symmetric\_difference function on B
5. >>> B.symmetric\_difference(A)
6. {1, 2, 3, 6, 7, 8}

## Different Python Set Methods

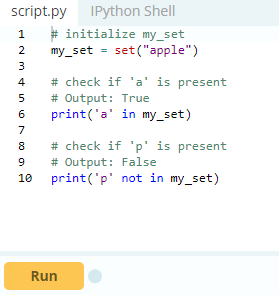
There are many set methods, some of which we have already used above. Here is a list of all the methods that are available with set objects.

|  |  |
| --- | --- |
| Method | Description |
| [add()](https://www.programiz.com/python-programming/methods/set/add) | Adds an element to the set |
| [clear()](https://www.programiz.com/python-programming/methods/set/clear) | Removes all elements from the set |
| [copy()](https://www.programiz.com/python-programming/methods/set/copy) | Returns a copy of the set |
| [difference()](https://www.programiz.com/python-programming/methods/set/difference) | Returns the difference of two or more sets as a new set |
| [difference\_update()](https://www.programiz.com/python-programming/methods/set/difference_update) | Removes all elements of another set from this set |
| [discard()](https://www.programiz.com/python-programming/methods/set/discard) | Removes an element from the set if it is a member. (Do nothing if the element is not in set) |
| [intersection()](https://www.programiz.com/python-programming/methods/set/intersection) | Returns the intersection of two sets as a new set |
| [intersection\_update()](https://www.programiz.com/python-programming/methods/set/intersection_update) | Updates the set with the intersection of itself and another |
| [isdisjoint()](https://www.programiz.com/python-programming/methods/set/isdisjoint) | Returns True if two sets have a null intersection |
| [issubset()](https://www.programiz.com/python-programming/methods/set/issubset) | Returns True if another set contains this set |
| [issuperset()](https://www.programiz.com/python-programming/methods/set/issuperset) | Returns True if this set contains another set |
| [pop()](https://www.programiz.com/python-programming/methods/set/pop) | Removes and returns an arbitary set element. Raise KeyError if the set is empty |
| [remove()](https://www.programiz.com/python-programming/methods/set/remove) | Removes an element from the set. If the element is not a member, raise a KeyError |
| [symmetric\_difference()](https://www.programiz.com/python-programming/methods/set/symmetric_difference) | Returns the symmetric difference of two sets as a new set |
| [symmetric\_difference\_update()](https://www.programiz.com/python-programming/methods/set/symmetric_difference_update) | Updates a set with the symmetric difference of itself and another |
| [union()](https://www.programiz.com/python-programming/methods/set/union) | Returns the union of sets in a new set |
| [update()](https://www.programiz.com/python-programming/methods/set/update) | Updates the set with the union of itself and others |
| Python Set Methods | |

## Other Set Operations

### Set Membership Test

We can test if an item exists in a set or not, using the keyword in.



### Iterating Through a Set

Using a for loop, we can iterate though each item in a set.

1. >>> for letter in set("apple"):
2. ... print(letter)
3. ...
4. a
5. p
6. e
7. 1

### Built-in Functions with Set

Built-in functions like all(), any(), enumerate(), len(), max(), min(), sorted(), sum() etc. are commonly used with set to perform different tasks.

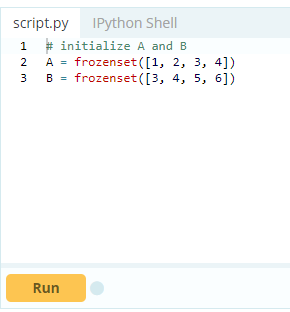
|  |  |
| --- | --- |
| Function | Description |
| [all()](https://www.programiz.com/python-programming/methods/built-in/all) | Return True if all elements of the set are true (or if the set is empty). |
| [any()](https://www.programiz.com/python-programming/methods/built-in/any) | Return True if any element of the set is true. If the set is empty, return False. |
| [enumerate()](https://www.programiz.com/python-programming/methods/built-in/enumerate) | Return an enumerate object. It contains the index and value of all the items of set as a pair. |
| [len()](https://www.programiz.com/python-programming/methods/built-in/len) | Return the length (the number of items) in the set. |
| [max()](https://www.programiz.com/python-programming/methods/built-in/max) | Return the largest item in the set. |
| [min()](https://www.programiz.com/python-programming/methods/built-in/min) | Return the smallest item in the set. |
| [sorted()](https://www.programiz.com/python-programming/methods/built-in/sorted) | Return a new sorted list from elements in the set(does not sort the set itself). |
| [sum()](https://www.programiz.com/python-programming/methods/built-in/sum) | Retrun the sum of all elements in the set. |
| Built-in Functions with Set | |

## 

## Python Frozenset

Frozenset is a new class that has the characteristics of a set, but its elements cannot be changed once assigned. While tuples are immutable lists, frozensets are immutable sets. Sets being mutable are unhashable, so they can't be used as dictionary keys. On the other hand, frozensets are hashable and can be used as keys to a dictionary. Frozensets can be created using the function [frozenset()](https://www.programiz.com/python-programming/methods/built-in/frozenset).

This datatype supports methods like copy(), difference(), intersection(), isdisjoint(), issubset(), issuperset(), symmetric\_difference() and union(). Being immutable it does not have method that add or remove elements.



Try these examples on Python shell.

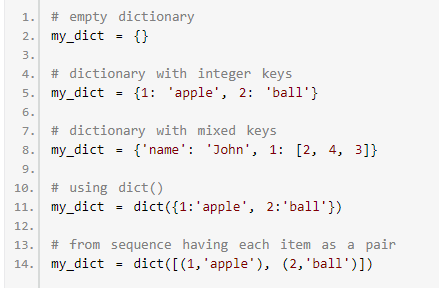
1. >>> A.isdisjoint(B)
2. False
3. >>> A.difference(B)
4. frozenset({1, 2})
5. >>> A | B
6. frozenset({1, 2, 3, 4, 5, 6})
7. >>> A.add(3)
8. ...
9. AttributeError: 'frozenset' object has no attribute 'add’

# **Python Dictionary**

Python dictionary is an unordered collection of items. While other compound data types have only value as an element, a dictionary has a key: value pair. Dictionaries are optimized to retrieve values when the key is known.

## How to create a dictionary?

Creating a dictionary is as simple as placing items inside curly braces {} separated by comma. An item has a key and the corresponding value expressed as a pair, key: value.While values can be of any data type and can repeat, keys must be of immutable type ([string](https://www.programiz.com/python-programming/string), [number](https://www.programiz.com/python-programming/numbers) or [tuple](https://www.programiz.com/python-programming/tuple) with immutable elements) and must be unique.

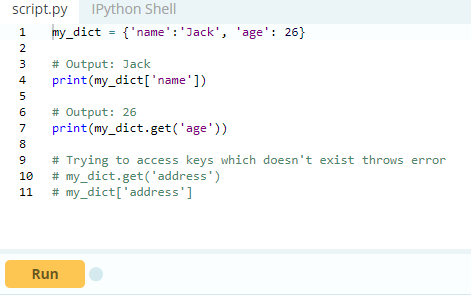


As you can see above, we can also create a dictionary using the built-in function dict().

## How to access elements from a dictionary?

While indexing is used with other container types to access values, dictionary uses keys. Key can be used either inside square brackets or with the get() method.

The difference while using get() is that it returns None instead of KeyError, if the key is not found.



When you run the program, the output will be:

Jack

26

How to change or add elements in a dictionary?

Dictionary are mutable. We can add new items or change the value of existing items using assignment operator. If the key is already present, value gets updated, else a new key: value pair is added to the dictionary.



When you run the program, the output will be:

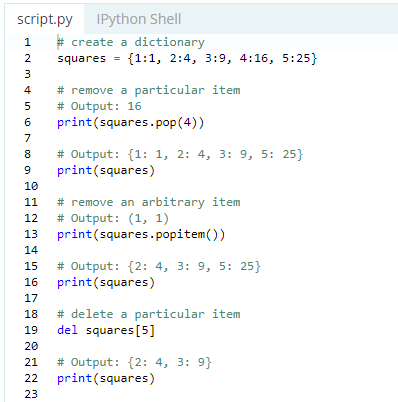
{'name': 'Jack', 'age': 27}

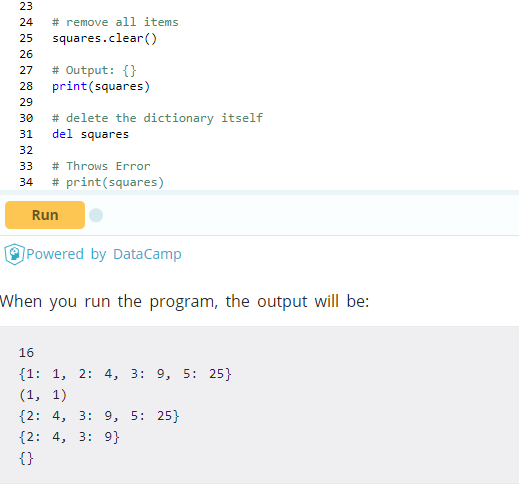
{'name': 'Jack', 'age': 27, 'address': 'Downtown'}

## How to delete or remove elements from a dictionary?

We can remove a particular item in a dictionary by using the method pop(). This method removes as item with the provided key and returns the value.

The method, popitem() can be used to remove and return an arbitrary item (key, value) form the dictionary. All the items can be removed at once using the clear() method. We can also use the del keyword to remove individual items or the entire dictionary itself.



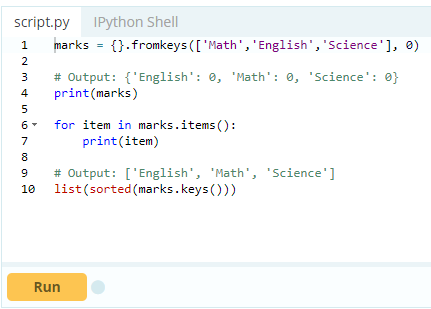


## Python Dictionary Methods

Methods that are available with dictionary are tabulated below. Some of them have already been used in the above examples.

|  |  |
| --- | --- |
| Method | Description |
| [clear()](https://www.programiz.com/python-programming/methods/dictionary/clear) | Remove all items form the dictionary. |
| [copy()](https://www.programiz.com/python-programming/methods/dictionary/copy) | Return a shallow copy of the dictionary. |
| [fromkeys(seq[, v])](https://www.programiz.com/python-programming/methods/dictionary/fromkeys) | Return a new dictionary with keys from seq and value equal to v(defaults to None). |
| [get(key[,d])](https://www.programiz.com/python-programming/methods/dictionary/get) | Return the value of key. If key doesnot exit, return d (defaults to None). |
| [items()](https://www.programiz.com/python-programming/methods/dictionary/items) | Return a new view of the dictionary's items (key, value). |
| [keys()](https://www.programiz.com/python-programming/methods/dictionary/keys) | Return a new view of the dictionary's keys. |
| [pop(key[,d])](https://www.programiz.com/python-programming/methods/dictionary/pop) | Remove the item with key and return its value or d if key is not found. If d is not provided and key is not found, raises KeyError. |
| [popitem()](https://www.programiz.com/python-programming/methods/dictionary/popitem) | Remove and return an arbitary item (key, value). Raises KeyError if the dictionary is empty. |
| [setdefault(key[,d])](https://www.programiz.com/python-programming/methods/dictionary/setdefault) | If key is in the dictionary, return its value. If not, insert key with a value of d and return d (defaults to None). |
| [update([other])](https://www.programiz.com/python-programming/methods/dictionary/update) | Update the dictionary with the key/value pairs from other, overwriting existing keys. |
| [values()](https://www.programiz.com/python-programming/methods/dictionary/values) | Return a new view of the dictionary's values |
| Python Dictionary Methods | |

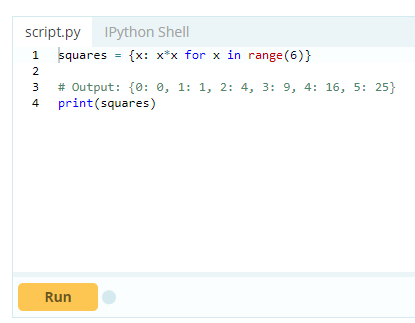
Here are a few example use of these methods.



## 

## Python Dictionary Comprehension

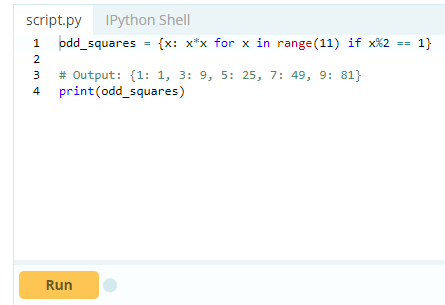
Dictionary comprehension is an elegant and concise way to create new dictionary from an iterable in Python. Dictionary comprehension consists of an expression pair (key: value) followed by forstatement inside curly braces {}. Here is an example to make a dictionary with each item being a pair of a number and its square.



This code is equivalent to

1. squares = {}
2. for x in range(6):
3. squares[x] = x\*x

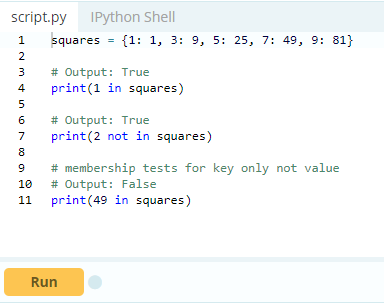
A dictionary comprehension can optionally contain more [for](https://www.programiz.com/python-programming/for-loop) or [if statements](https://www.programiz.com/python-programming/if-elif-else). An optional if statement can filter out items to form the new dictionary. Here are some examples to make dictionary with only odd items.



## Other Dictionary Operations

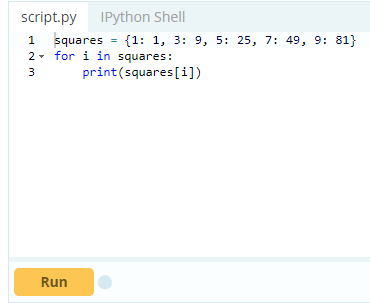
### Dictionary Membership Test

We can test if a key is in a dictionary or not using the keyword in. Notice that membership test is for keys only, not for values.



### Iterating Through a Dictionary

Using a for loop we can iterate though each key in a dictionary.



### Built-in Functions with Dictionary

Built-in functions like all(), any(), len(), cmp(), sorted() etc. are commonly used with dictionary to perform different tasks.

|  |  |
| --- | --- |
| Function | Description |
| [all()](https://www.programiz.com/python-programming/methods/built-in/all) | Return True if all keys of the dictionary are true (or if the dictionary is empty). |
| [any()](https://www.programiz.com/python-programming/methods/built-in/any) | Return True if any key of the dictionary is true. If the dictionary is empty, return False. |
| [len()](https://www.programiz.com/python-programming/methods/built-in/len) | Return the length (the number of items) in the dictionary. |
| cmp() | Compares items of two dictionaries. |
| [sorted()](https://www.programiz.com/python-programming/methods/built-in/sorted) | Return a new sorted list of keys in the dictionary. |
| Built-in Functions with Dictionary | |

Here are some examples that uses built-in functions to work with dictionary.

# 

# **Python Nested Dictionary**

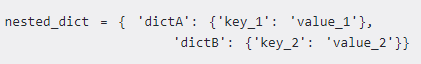
In Python, a dictionary is an unordered collection of items. For example:



Here, dictionary has a key:value pair enclosed within curly brackets {}. To learn more about dictionary, please visit [Python Dictionary](https://www.programiz.com/python-programming/dictionary).

## What is Nested Dictionary in Python?

In Python, a nested dictionary is a dictionary inside a dictionary. It's a collection of dictionaries into one single dictionary.



Here, the nested\_dict is a nested dictionary with the dictionary dictA and dictB. They are two dictionary each having own key and value.

## Create a Nested Dictionary

We're going to create dictionary of people within a dictionary.

### Example 1: How to create a nested dictionary

### 

### When we run above program, it will output:

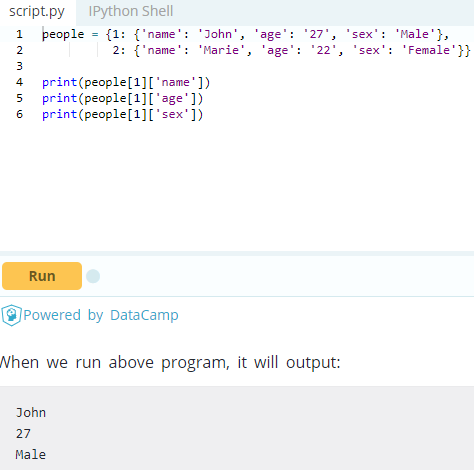
{1: {'name': 'John', 'age': '27', 'sex': 'Male'}, 2: {'name': 'Marie', 'age': '22', 'sex': 'Female'}}

In the above program, people is a nested dictionary. The internal dictionary 1 and 2 is assigned to people. Here, both the dictionary have key name, age , sex with different values. Now, we print the result of people.

## Access elements of a Nested Dictionary

To access element of a nested dictionary, we use indexing [] syntax in Python.

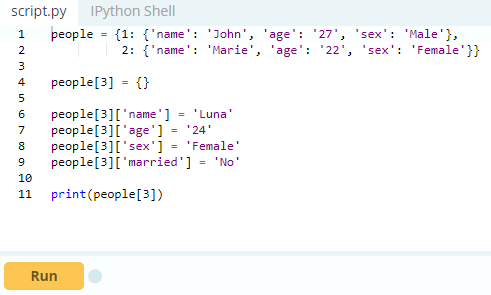
### Example 2: Access the elements using the [] syntax



In the above program, we print the value of key name using i.e. people[1]['name'] from internal dictionary 1. Similarly, we print the value of age and sex one by one.

## Add element to a Nested Dictionary

### Example 3: How to change or add elements in a nested dictionary?

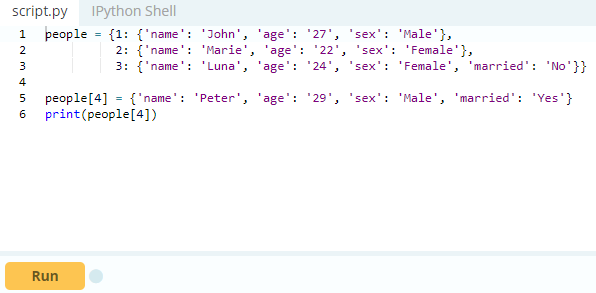


When we run above program, it will output:

{'name': 'Luna', 'age': '24', 'sex': 'Female', 'married': 'No'}

In the above program, we create an empty dictionary 3 inside the dictionary people. Then, we add the key:value pair i.e people[3]['Name'] = 'Luna' inside the dictionary 3. Similarly, we do this for key age, sex and married one by one. When we print the people[3], we get key:value pairs of dictionary 3.

### Example 4: Add another dictionary to the nested dictionary



When we run above program, it will output:

{'name': 'Peter', 'age': '29', 'sex': 'Male', 'married': 'Yes'}

In the above program, we assign a dictionary literal to people[4]. The literal have keys name, age and sex with respective values. Then we print the people[4], to see that the dictionary 4 is added in nested dictionary people.

## Delete elements from a Nested Dictionary

In Python, we use “del “ statement to delete elements from nested dictionary.

### Example 5: How to delete elements from a nested dictionary?



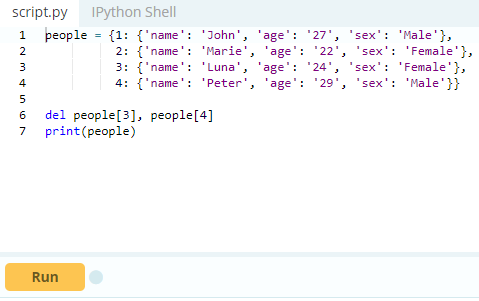
When we run above program, it will output:

{'name': 'Luna', 'age': '24', 'sex': 'Female'}

{'name': 'Peter', 'age': '29', 'sex': 'Male'}

In the above program, we delete the key:value pairs of married from internal dictionary 3and 4. Then, we print the people[3] and people[4] to confirm changes.

### Example 6: How to delete dictionary from a nested dictionary?



When we run above program, it will output:

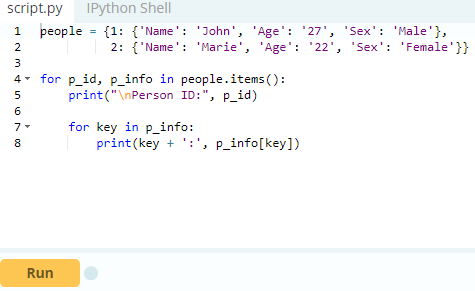
{1: {'name': 'John', 'age': '27', 'sex': 'Male'}, 2: {'name': 'Marie', 'age': '22', 'sex': 'Female'}}

In the above program, we delete both the internal dictionary 3 and 4 using del from the nested dictionary people. Then, we print the nested dictionary people to confirm changes.

## Iterating Through a Nested Dictionary

Using the for loops, we can iterate through each elements in a nested dictionary.

### Example 7: How to iterate through a Nested dictionary?



When we run above program, it will output:

Person ID: 1

Name: John

Age: 27

Sex: Male

Person ID: 2

Name: Marie

Age: 22

Sex: Female

In the above program, the first loop returns all the keys in the nested dictionary people. It consist of the IDs p\_id of each person. We use these IDs to unpack the information p\_infoof each person.

The second loop goes through the information of each person. Then, it returns all of the keys name, age, sex of each person's dictionary. Now, we print the key of the person’s information and the value for that key.

## Key Points to Remember:

1. Nested dictionary is an unordered collection of dictionary
2. Slicing Nested Dictionary is not possible.
3. We can shrink or grow nested dictionary as need.
4. Like Dictionary, it also has key and value.
5. Dictionary are accessed using key.

# **Python Arrays**

In programming, an array is a collection of elements of the same type. Arrays are popular in most programming languages like Java, C/C++, JavaScript and so on. However, in Python, they are not that common. When people talk about Python arrays, more often than not, they are talking about Python lists. If you don't know what lists are, you should definitely check [Python list](https://www.programiz.com/python-programming/list) article. That being said, array of numeric values are supported in Python by the [array module](https://docs.python.org/3/library/array.html).

## Python Lists Vs array Module as Arrays

We can treat lists as arrays. However, we cannot constrain the type of elements stored in a list. For example:

1. a = [1, 3.5, "Hello"]

If you create arrays using the array module, all elements of the array must be of the same numeric type.

import array as arr

a = arr.array('d', [1, 3.5, "Hello"]) // Error

## How to create arrays?

As you might have guessed from the above example, we need to import array module to create arrays. For example:

1. import array as arr
2. a = arr.array('d', [1.1, 3.5, 4.5])
3. print(a)

Here, we created an array of float type. The letter 'd' is a type code. This determines the type of the array during creation.

**Commonly used type codes:**

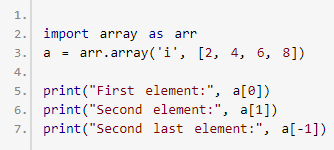
| Code | C Type | Python Type | Min bytes |
| --- | --- | --- | --- |
| 'b' | signed char | int | 1 |
| 'B' | unsigned char | int | 1 |
| 'u' | Py\_UNICODE | Unicode | 2 |
| 'h' | signed short | int | 2 |
| 'H' | unsigned short | int | 2 |
| 'i' | signed int | int | 2 |
| 'I' | unsigned int | int | 2 |
| 'l' | signed long | int | 4 |
| 'L' | unsigned long | int | 4 |
| 'f' | float | float | 4 |
| 'd' | double | float | 8 |

We will not discuss about different C types in this article. We will use two type codes in this entire article: 'i' for integers and  'd' for floats.

Note: The 'u' type code for Unicode characters is deprecated since version 3.3. Avoid using it when possible.

## How to access array elements?

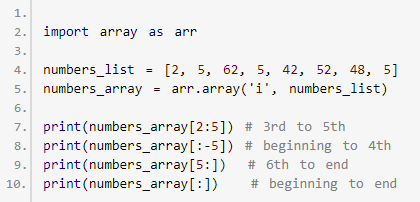
We use indices to access elements of an array:



Remember, the index starts from 0 (not 1) similar to lists.

## How to slice arrays?

We can access a range of items in an array by using the slicing operator :



When you run the program, the output will be:

array('i', [62, 5, 42])

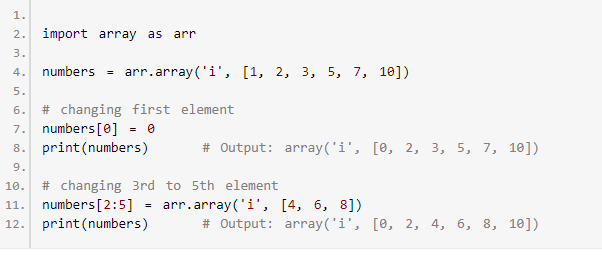
array('i', [2, 5, 62])

array('i', [52, 48, 5])

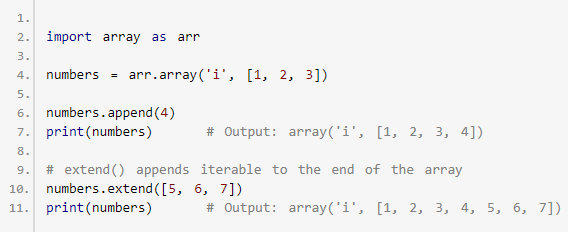
array('i', [2, 5, 62, 5, 42, 52, 48, 5])

## How to change or add elements?

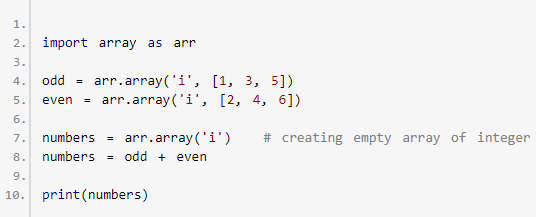
Arrays are mutable; their elements can be changed in a similar way like lists.



We can add one item to a list using the append() method, or add several items using extend() method

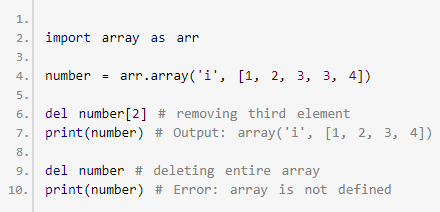


We can concatenate two arrays using + operator.

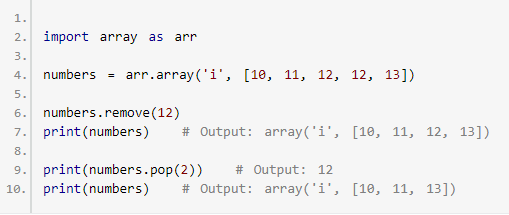


## How to remove/delete elements?

We can delete one or more items from an array using [Python's del statement](https://www.programiz.com/python-programming/del).



We can use the remove() method to remove the given item, and pop() method to remove an item at the given index.

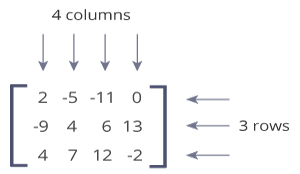


## When to use arrays?

Lists are much more flexible than arrays. They can store elements of different data types including string. Also, lists are faster than arrays. And, if you need to do mathematical computation on arrays and matrices, you are much better off using something like [NumPy](https://www.programiz.com/python-programming/matrix#numpy)library. Unless you don't really need arrays (array module may be needed to interface with C code), don't use them.

# **Python Matrices and NumPy Arrays**

A matrix is a two-dimensional data structure where numbers are arranged into rows and columns. For example:



This matrix is a 3x4 (pronounced "three by four") matrix because it has 3 rows and 4 columns.

Python Matrix

Python doesn't have a built-in type for matrices. However, we can treat list of a list as a matrix. For example:

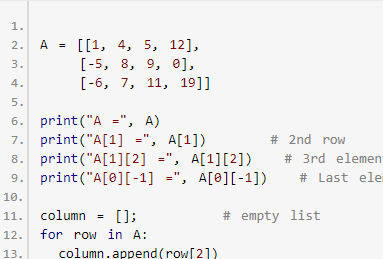
1. A = [[1, 4, 5],
2. [-5, 8, 9]]

We can treat this list of a list as a matrix having 2 rows and 3 columns.



Be sure to learn about [Python lists](https://www.programiz.com/python-programming/list) before proceed this article.

Let's see how to work with a nested list.



When we run the program, the output will be:

A = [[1, 4, 5, 12], [-5, 8, 9, 0], [-6, 7, 11, 19]]

A[1] = [-5, 8, 9, 0]

A[1][2] = 9

A[0][-1] = 12

3rd column = [5, 9, 11]

Here are few more examples related to Python matrices using nested lists.

* [Add two matrices](https://www.programiz.com/python-programming/examples/add-matrix)
* [Transpose a Matrix](https://www.programiz.com/python-programming/examples/transpose-matrix)
* [Multiply two matrices](https://www.programiz.com/python-programming/examples/multiply-matrix)

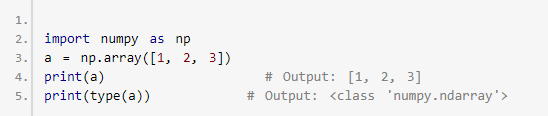
Using nested lists as a matrix works for simple computational tasks, however, there is a better way of working with matrices in Python using [NumPy](https://docs.scipy.org/doc/numpy-1.10.1/user/whatisnumpy.html) package.

## NumPy Array

NumPy is a package for scientific computing which has support for a powerful N-dimensional array object. Before you can use NumPy, you need to install it. For more info,

* Visit: [How to install NumPy?](https://scipy.org/install.html)
* If you are on Windows, download and install [anaconda distribution](https://docs.anaconda.com/anaconda/install/windows/) of Python. It comes with NumPy and other several packages related to data science and machine learning.

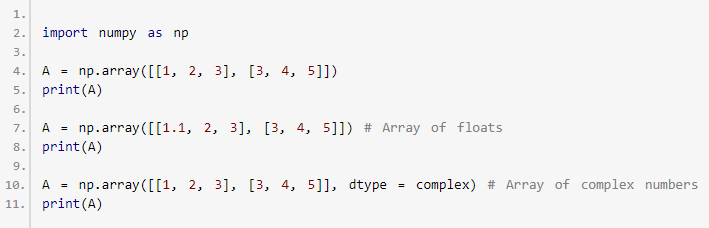
Once NumPy is installed, you can import and use it. NumPy provides multidimensional array of numbers (which is actually an object). Let's take an example:



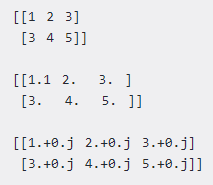
## How to create a NumPy array?

There are several ways to create NumPy arrays.

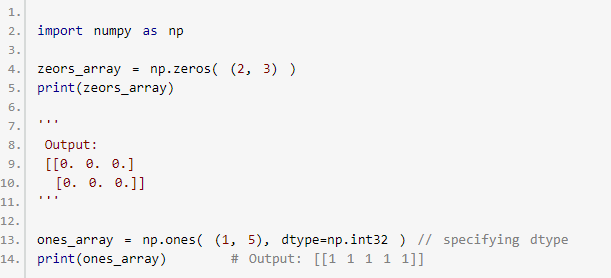
**1. Array of integers, floats and complex Numbers**



When you run the program, the output will be:

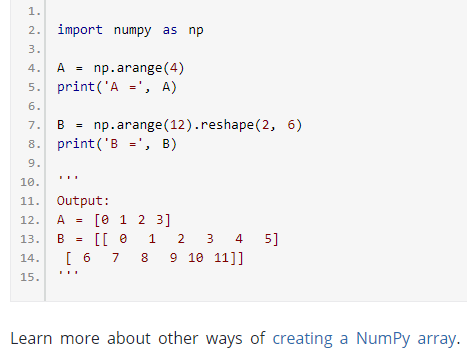


**2. Array of zeros and ones**



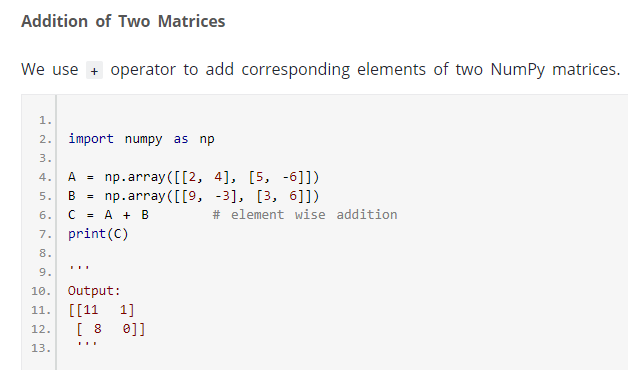
Here, we have specified dtype to 32 bits (4 bytes). Hence, this array can take values from -2-31 to 2-31-1.

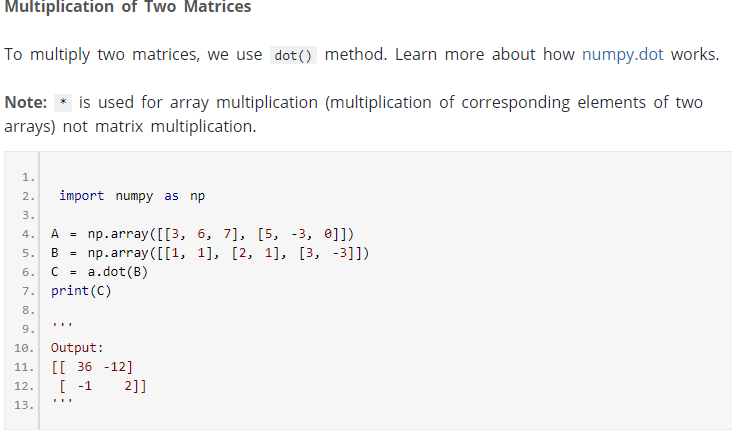
**3. Using arange() and shape()**

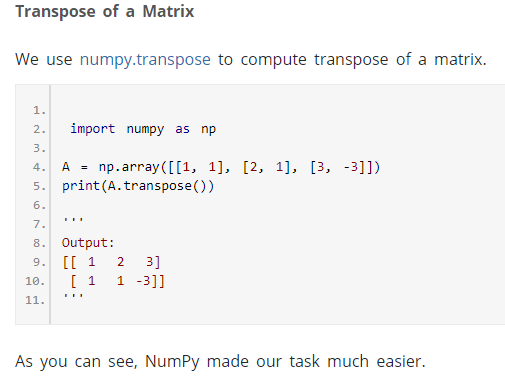


## Matrix Operations

Above, we gave you 3 examples: addition of two matrices, multiplication of two matrices and transpose of a matrix. We used nested lists before to write those programs. Let's see how we can do the same task using NumPy array.



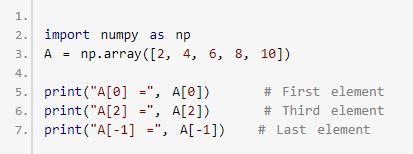




## Access matrix elements, rows and columns

**Access matrix elements**

Similar like lists, we can access matrix elements using index. Let's start with a one-dimensional NumPy array.



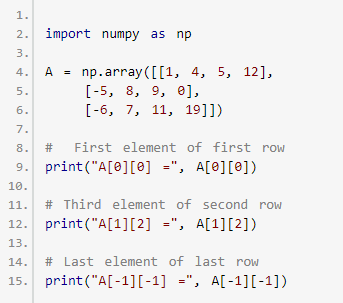
When you run the program, the output will be:

A[0] = 2

A[2] = 6

A[-1] = 10

Now, let's see how we can access elements of a two-dimensional array (which is basically a matrix).



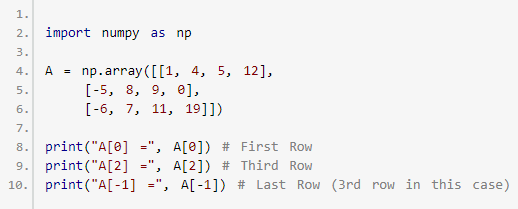
When we run the program, the output will be:

A[0][0] = 1

A[1][2] = 9

A[-1][-1] = 19

**Access rows of a Matrix**



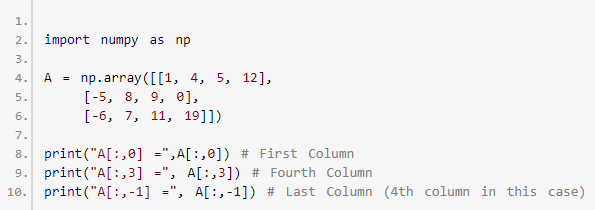
When we run the program, the output will be:

A[0] = [1, 4, 5, 12]

A[2] = [-6, 7, 11, 19]

A[-1] = [-6, 7, 11, 19]

**Access columns of a Matrix**



When we run the program, the output will be:

A[:,0] = [ 1 -5 -6]

A[:,3] = [12 0 19]

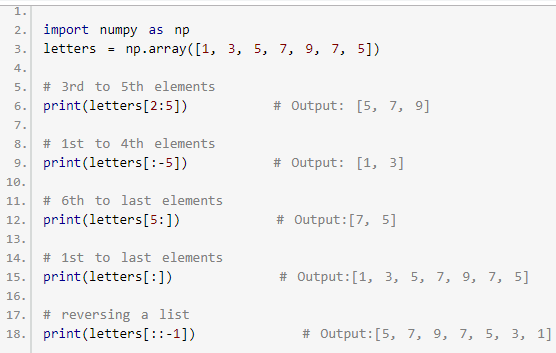
A[:,-1] = [12 0 19]

If you don't know how this above code works, read slicing of a matrix section of this article.

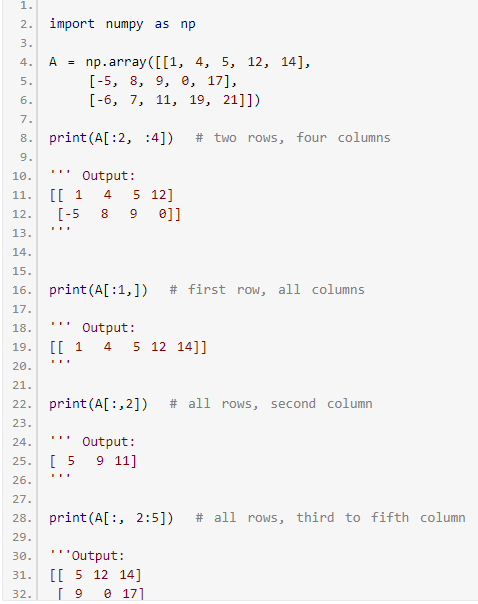
## Slicing of a Matrix

Slicing of a one-dimensional NumPy array is similar to a list. If you don't know how slicing for a list works, visit [Understanding Python's slice notation](https://stackoverflow.com/questions/509211/understanding-pythons-slice-notation).

Let's take an example:



Now, let's see how we can slice a matrix.



33. [11 19 21]

34 …

As you can see, using NumPy (instead of nested lists) makes it a lot easier to work with matrices, and we haven't even scratched the basics. We suggest you to explore NumPy package in detail especially if you trying to use Python for data science/analytics.

**NumPy Resources you might might helpful:**

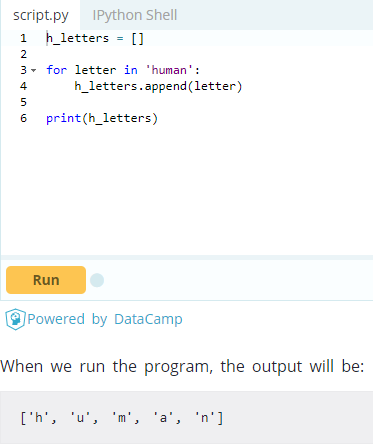
* [Quickstart NumPy tutorial](https://docs.scipy.org/doc/numpy-1.15.1/user/quickstart.html)
* [NumPy tutorial](http://www.labri.fr/perso/nrougier/teaching/numpy/numpy.html)
* [NumPy Reference](https://docs.scipy.org/doc/numpy-1.14.5/reference/)

# **Python List Comprehension**

## List Comprehension vs For Loop in Python

Suppose, we want to separate the letters of the word human and add the letters as items of a list. The first thing that comes in mind would be using [for loop](https://www.programiz.com/python-programming/for-loop).

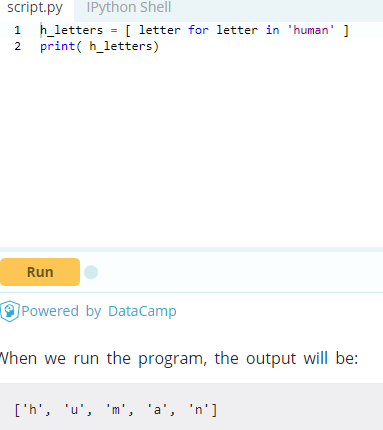
#### **Example 1: Iterating through a string Using for Loop**



However, Python has an easier way to solve this issue using List Comprehension. List comprehension is an elegant way to define and create lists based on existing lists.

Let’s see how the above program can be written using list comprehensions.

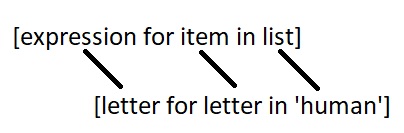
### Example 2: Iterating through a string Using List Comprehension



In the above example, a new list is assigned to variable h\_letters, and list contains the items of the iterable string 'human'. We call print() function to receive the output.

### Syntax of List Comprehension

[expression for item in list]



We can now identify where list comprehensions are used. If you noticed, human is a string, not a list. This is the power of list comprehension. It can identify when it receives a string or a tuple and work on it like a [list](https://www.programiz.com/python-programming/list). You can do that using loops. However, not every loop can be rewritten as list comprehension. But as you learn and get comfortable with list comprehensions, you will find yourself replacing more and more loops with this elegant syntax.

## List Comprehensions vs Lambda functions

List comprehensions aren’t the only way to work on lists. Various built-in functions and [lambda functions](https://www.programiz.com/python-programming/anonymous-function) can create and modify lists in less lines of code.

#### **Example 3: Using Lambda functions inside List**

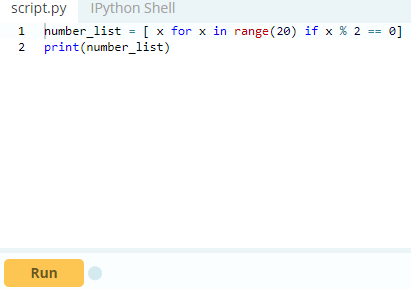


However, list comprehensions are usually more human readable than lambda functions. It is easier to understand what the programmer was trying to accomplish when list comprehensions are used.

## Conditionals in List Comprehension

List comprehensions can utilize conditional statement to modify existing list (or other tuples). We will create list that uses mathematical operators, integers, and [range()](https://www.programiz.com/python-programming/methods/built-in/range).

#### **Example 4: Using if with List Comprehension**

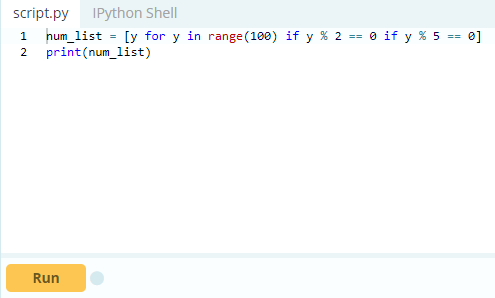
****

When we run the above program, the output will be:

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

The list ,number\_list, will be populated by the items in range from 0-19 if the item's value is divisible by 2.

#### **Example 5: Nested IF with List Comprehension**



When we run the above program, the output will be:

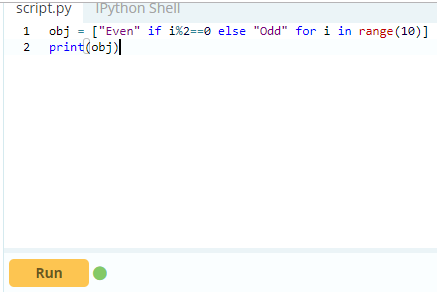
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]

Here, list comprehension checks:

1. Is y divisible by 2 or not?
2. Is y divisible by 5 or not?

If y satisfies both conditions, y is appended to num\_list.

#### **Example 6: if...else With List Comprehension**



When we run the above program, the output will be:

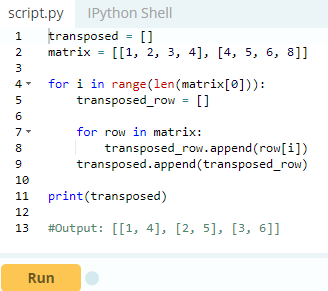
['Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd']

Here, list comprehension will check the 10 numbers from 0 to 9. If i is divisible by 2, then Even is appended to the obj list. If not, Odd is appended.

## Nested Loops in List Comprehension

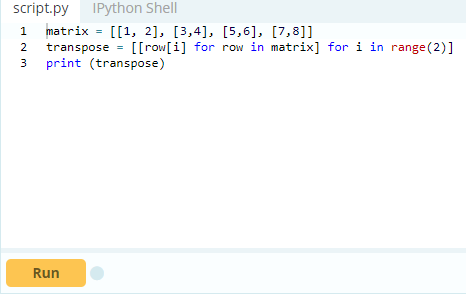
Suppose, we need to compute transpose of a matrix which requires nested for loop. Let’s see how it is done using normal for loop first.

#### **Example 7: Transpose of Matrix using Nested Loops**



The above code use two for loops to find transpose of the matrix. We can also perform nested iteration inside a list comprehension. In this section, we will find transpose of a matrix using nested loop inside list comprehension.

#### **Example 8: Transpose of a Matrix using List Comprehension**



When we run the above program, the output will be:

[[1, 3, 5, 7], [2, 4, 6, 8]]

In above program, we have a variable matrix which have 4 rows and 2 columns. We need to find transpose of the matrix. For that, we used list comprehension.

**\*\*Note:** The nested loops in list comprehension don’t work like normal nested loops. In the above program, for i in range(2) is executed before row[i] for row in matrix. Hence at first, a value is assigned to i then item directed by row[i] is appended in the transpose variable.

## Key Points to Remember

* List comprehension is an elegant way to define and create lists based on existing lists.
* List comprehension is generally more compact and faster than normal functions and loops for creating list.
* However, we should avoid writing very long list comprehensions in one line to ensure that code is user-friendly.
* Remember, every list comprehension can be rewritten in for loop, but every for loop can’t be rewritten in the form of list comprehension.