**Build-in Data Types**

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 and unindexed. No duplicate members.  
Dictionary is a collection which is unordered and changeable. No duplicate members.

**LIST ITEMS**

List items are ordered, changeable, and allow duplicate values.  
The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.  
Since lists are indexed, lists can have items with the same value.

**ACCESS LIST ITEMS**

fruits = ["apple", "banana", "cherry", "apple", "cherry", "peach", "orange",]

print(len(fruits)) = To determine how many items a list has, use the len() function.

print(fruits[-1]) = -1 refers to the last item, -2 refers to the second last item etc.

print(fruits[2:5]) = Return the third, fourth, and fifth item.

print(fruits[:4]) = This example returns the items from the beginning to, but NOT including, "apple".

print(fruits[2:]) = This example returns the items from "cherry" to the end.

print( [-4:-1]) = This example returns the items from "apple" (-4) to, but NOT including "orange".

**CHANGE LIST ITEMS**

fruits[1] = "bananas" = Change the second item.

fruits[1:3] = ["bananas", " cherries"] = Change the second and the third items.

fruits[1:2] = ["bananas", " cherries"] = Change the second value by replacing it with two new values.

fruits[1:3] = ["bananas"] = Change the second and the third values by replacing it with new value.

**ADD LIST ITEMS**

fruits.insert(2, "watermelon") = To insert a new list item, without replacing any of the existing values, we can use the insert() method.

fruits.append("orange") = To add an item to the end of the list, use the append() method.

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

fruits.extend(tropical) = To append elements from another list to the current list, use the extend() method.

The extend() method does not have to append lists, you can add any iterable object (tuples, sets, dictionaries etc.).

**REMOVE LIST ITEMS**

fruits.remove("banana") = The remove() method removes the specified item.

fruits.pop(1) = The pop() method removes the specified index.

fruits.pop() = Remove the last item.

del fruits[0] = The del keyword also removes the specified index.

del fruits = The del keyword can also delete the list completely.

fruits.clear() = The clear() method empties the list. The list still remains, but it has no content.

**LOOP LISTS**

for x in thislist:  
  print(x)

for i in range(len(thislist)):  
  print(thislist[i])

i = 0  
while i < len(thislist):  
  print(thislist[i])  
  i = i + 1

**LIST COMPREHENSION**

[print(x) for x in thislist]

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name:

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

The return value is a new list, leaving the old list unchanged:

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

Set the values in the new list to upper case:

newlist = [x.upper() for x in fruits]

Return "orange" instead of "banana":

newlist = [x if x != "banana" else "orange" for x in fruits]

You can use the range() function to create an iterable:

newlist = [x for x in range(10)]

Accept only numbers lower than 5:

newlist = [x for x in range(10) if x < 5]

**SORT LIST**

Sort the list alphabetically: fruits.sort()

Sort the list descending: fruits.sort(reverse = True)

By default the sort() method is case sensitive, resulting in all capital letters being sorted before lower case letters. Luckily we can use built-in functions as key functions when sorting a list.  
So if you want a case-insensitive sort function, use str.lower as a key function:

fruits.sort(key = str.lower)

fruits.reverse() = The reverse() method reverses the current sorting order of the elements.

**COPY LIST**

Make a copy of a list with the copy() method: mylist = fruits.copy()

Make a copy of a list with the list() method: mylist = list(fruits)

**JOIN LIST**

One of the easiest ways are by using the + operator: list3 = list1 + list2

Another way to join two lists are by appending all the items from list2 into list1, one by one:

for x in list2:  
  list1.append(x)

Or you can use the extend() method, which purpose is to add elements from one list to another list:

list1.extend(list2)

**LIST METHODS**

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() Sorts the list

**TUPLE ITEMS**

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

To determine how many items a tuple has, use the len() function: print(len(thistuple))

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",)

**UPDATE TUPLES**

You can convert the tuple into a list, change the list, and convert the list back into a tuple.

y = list(thistuple)  
y[1] = "kiwi"  
thistuple = tuple(y)

Convert the tuple into a list, add "orange", and convert it back into a tuple:

y = list(thistuple)  
y.append("orange")

Convert the tuple into a list, remove "apple", and convert it back into a tuple:

y = list(thistuple)  
y.remove("apple")  
thistuple = tuple(y)

The del keyword can delete the tuple completely: del thistuple

**UNPACK TUPLES**

When we create a tuple, we normally assign values to it. This is called "packing" a tuple:

fruits = ("apple", "banana", "cherry")

But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking":

fruits = ("apple", "banana", "cherry")  
(green, yellow, red) = fruits  
print(green)  
print(yellow)  
print(red)

If the number of variables is less than the number of values, you can add an \* to the variable name and the values will be assigned to the variable as a list:

fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")  
(green, yellow, \*red) = fruits  
print(green)  
print(yellow)  
print(red)

If the asterix is added to another variable name than the last, Python will assign values to the variable until the number of values left matches the number of variables left.

fruits = ("apple", "mango", "papaya", "pineapple", "cherry")  
(green, \*tropic, red) = fruits  
print(green)  
print(tropic)  
print(red)

**TUPLE METHODS**

count() Returns the number of times a specified value occurs in a tuple  
index() Searches the tuple for a specified value and returns the position of where it was found

**SET ITEMS**

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

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

Using the set() constructor to make a set: thisset = set(("apple", "banana", "cherry"))

**ACCESS SET ITEMS**

for x in thisset:  
  print(x)

**ADD SET ITEMS**

To add one item to a set use the add() method: thisset.add("orange")

To add items from another set into the current set, use the update() method:

tropical = {"pineapple", "mango", "papaya"}

thisset.update(tropical)

The object in the update() method does not have be a set, it can be any iterable object (tuples, lists, dictionaries etc.): mylist = ["kiwi", "orange"]

thisset.update(mylist)

**REMOVE SET ITEMS**

To remove an item in a set, use the remove(), or the discard() method.

thisset.remove("banana")

**Note:** If the item to remove does not exist, remove() will raise an error.

thisset.discard("banana")

**Note:** If the item to remove does not exist, discard() will **NOT** raise an error.

You can also use the pop() method to remove an item, but this method will remove the last item. Remember that sets are unordered, so you will not know what item that gets removed. The return value of the pop() method is the removed item.

The clear() method empties the set: thisset.clear()

The del keyword will delete the set completely: del thisset

**JOIN SETS**

The union() method returns a new set with all items from both sets: set3 = set1.union(set2)

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

**Note:** Both union() and update() will exclude any duplicate items.

**Keep ONLY the Duplicates**

The intersection\_update() method will keep only the items that are present in both sets.

x.intersection\_update(y)

The intersection() method will return a new set, that only contains the items that are present in both sets.

z = x.intersection(y)

The symmetric\_difference\_update() method will keep only the elements that are NOT present in both sets.

**Keep All, But NOT the Duplicates**

x.symmetric\_difference\_update(y)

The symmetric\_difference() method will return a new set, that contains only the elements that are NOT present in both sets.

z = x.symmetric\_difference(y)

**SET METHODS**

add() Adds an element to the set  
clear() Removes all the elements from the set  
copy() Returns a copy of the set  
difference() Returns a set containing the difference between two or more sets  
difference\_update() Removes the items in this set that are also included in another, specified set  
discard() Remove the specified item  
isdisjoint() Returns whether two sets have a intersection or not  
issubset() Returns whether another set contains this set or not  
issuperset() Returns whether this set contains another set or not  
pop() Removes an element from the set  
remove() Removes the specified element

**DICTIONARY ITEMS**

Dictionary items are ordered, changeable, and does not allow duplicates.  
Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

thisdict = {  
  "brand": "Toyota",  
  "model": "RAV-4",  
  "year": 2002  
 "colors": ["gray", "white", "blue"]  
}  
print(thisdict["brand"])

To determine how many items a dictionary has, use the len() function: print(len(thisdict))

**ACCESS ITEMS**

Get the value of the "model" key: x = thisdict["model"]

There is also a method called get() that will give you the same result: x = thisdict.get("model")

**Get Keys**

The keys() method will return a list of all the keys in the dictionary: x = thisdict.keys()

**Get Values**

The values() method will return a list of all the values in the dictionary: x = thisdict.values()

**Get Items**

The items() method will return each item in a dictionary, as tuples in a list: x = thisdict.items()

**CHANGE ITEMS**

You can change the value of a specific item by referring to its key name: thisdict["year"] = 2018

The update() method will update the dictionary with the items from the given argument.  
The argument must be a dictionary, or an iterable object with key:value pairs.

thisdict.update({"year": 2020})

**ADD ITEMS**

Add a color item to the dictionary by using the update() method: thisdict.update({"color": "red"})

**REMOVE ITEMS**

The pop() method removes the item with the specified key name: thisdict.pop("model")

The popitem() method removes the last inserted item (in versions before 3.7, a random item is removed instead): thisdict.popitem()

The del keyword removes the item with the specified key name: del thisdict["model"]

The del keyword can also delete the dictionary completely: del thisdict

The clear() method empties the dictionary: thisdict.clear()

**LOOP DICTIONARIES**

You can loop through a dictionary by using a for loop. When looping through a dictionary, the return value are the keys of the dictionary, but there are methods to return the values as well.

Print all key names in the dictionary, one by one:

for x in thisdict:  
  print(x)

Print all values in the dictionary, one by one:

for x in thisdict:  
  print(thisdict[x])

You can also use the values() method to return values of a dictionary:

for x in thisdict.values():  
  print(x)

You can use the keys() method to return the keys of a dictionary:

for x in thisdict.keys():  
  print(x)

Loop through both keys and values, by using the items() method:

for x, y in thisdict.items():  
  print(x, y)

**COPY DICTIONARIES**

Make a copy of a dictionary with the copy() method: mydict = thisdict.copy()

Make a copy of a dictionary with the dict() function: mydict = dict(thisdict)

**DICTIONARY METHODS**

fromkeys() Returns a dictionary with the specified keys and value  
get() Returns the value of the specified key  
items() Returns a list containing a tuple for each key value pair  
setdefault() Returns the value of the specified key. If the key does not exist: insert the key, with the specified value

**NESTED DICTIONARIES**

Create a dictionary that contain three dictionaries:

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

Create three dictionaries, then create one dictionary that will contain the other three dictionaries:

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

**PYTHON STRINGS**

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

a = "Hello, World!"  
print(a[1])

The upper() method returns the string in upper case: print(a.upper())

The lower() method returns the string in lower case: print(a.lower())

The strip() method removes any whitespace from the beginning or the end: print(a.strip())

The replace() method replaces a string with another string: print(a.replace("H", "J"))

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

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

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

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

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

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

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."

\' Single Quote   
\\ Backslash   
\n New Line   
\r Carriage Return   
\t Tab   
\b Backspace

**Method Description**

capitalize() Converts the first character to upper case  
casefold() Converts string into lower case  
center() Returns a centered string  
count() Returns the number of times a specified value occurs in a string  
encode() Returns an encoded version of the string  
endswith() Returns true if the string ends with the specified value  
expandtabs() Sets the tab size of the string  
find() Searches the string for a specified value and returns the position of where it was found  
format() Formats specified values in a string  
format\_map() Formats specified values in a string  
index() Searches the string for a specified value and returns the position of where it was found  
isalnum() Returns True if all characters in the string are alphanumeric  
isalpha() Returns True if all characters in the string are in the alphabet  
isdecimal() Returns True if all characters in the string are decimals  
isdigit() Returns True if all characters in the string are digits  
isidentifier() Returns True if the string is an identifier  
islower() Returns True if all characters in the string are lower case  
isnumeric() Returns True if all characters in the string are numeric  
isprintable() Returns True if all characters in the string are printable  
isspace() Returns True if all characters in the string are whitespaces  
istitle() Returns True if the string follows the rules of a title  
isupper() Returns True if all characters in the string are upper case  
join() Joins the elements of an iterable to the end of the string  
ljust() Returns a left justified version of the string  
lower() Converts a string into lower case  
lstrip() Returns a left trim version of the string  
maketrans() Returns a translation table to be used in translations  
partition() Returns a tuple where the string is parted into three parts  
rfind() Searches the string for a specified value and returns the last position of where it was found  
rindex() Searches the string for a specified value and returns the last position of where it was found  
rjust() Returns a right justified version of the string  
rpartition() Returns a tuple where the string is parted into three parts  
rsplit() Splits the string at the specified separator, and returns a list  
rstrip() Returns a right trim version of the string  
splitlines() Splits the string at line breaks and returns a list  
startswith() Returns true if the string starts with the specified value  
swapcase() Swaps cases, lower case becomes upper case and vice versa  
title() Converts the first character of each word to upper case  
translate() Returns a translated string  
upper() Converts a string into upper case  
zfill() Fills the string with a specified number of 0 values at the beginning

**PYTHON FUNCTIONS**

A function is a block of code which only runs when it is called.  
You can pass data, known as parameters, into a function.  
A function can return data as a result.

def my\_function():  
  print("Hello from a function")

**my\_function()**

**Arbitrary Arguments, \*args**

If you do not know how many arguments that will be passed into your function, add a \* before the parameter name in the function definition.  
This way the function will receive a tuple of arguments, and can access the items accordingly:

def my\_function(\*kids):  
  print("The youngest child is " + kids[2])  
  
my\_function("Emil", "Tobias", "Linus")

**Arbitrary Keyword Arguments, \*\*kwargs**

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: \*\* before the parameter name in the function definition.  
This way the function will receive a dictionary of arguments, and can access the items accordingly:

def my\_function(\*\*kid):  
  print("His last name is " + kid["lname"])  
  
my\_function(fname = "Tobias", lname = "Refsnes")

Default Parameter Value

If we call the function without argument, it uses the default value:

def my\_function(**country = "Norway"**):  
  print("I am from " + country)

To let a function return a value, use the return statement:

def my\_function(x):  
  **return 5 \* x**

**PYTHON LAMBDA**

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

Add 10 to argument a, and return the result: x = lambda a : a + 10

Multiply argument a with argument b and return the result: x = lambda a, b : a \* b

Summarize argument a, b, and c and return the result: x = lambda a, b, c : a + b + c

Use that function definition to make a function that always doubles the number you send in:

def myfunc(n):  
  return lambda a : a \* n  
  
mydoubler = myfunc(2)  
  
print(mydoubler(11))

**PYTHON CLASSES AND OBJECTS**

Create a class named MyClass, with a property named x:

class MyClass:  
  x = 5

Now we can use the class named MyClass to create objects:

p1 = MyClass()  
print(p1.x)

Create a class named Person, use the \_\_init\_\_() function to assign values for name and age:

class Person:  
  def \_\_init\_\_(self, name, age):  
    self.name = name  
    self.age = age  
  
p1 = Person("John", 36)  
  
print(p1.name)  
print(p1.age)

Objects can also contain methods. Methods in objects are functions that belong to the object.

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

You can modify properties on objects like this:  
p1.age = 40

Delete the age property from the p1 object:  
del p1.age

You can delete objects by using the del keyword:  
del p1

class definitions cannot be empty, but if you for some reason have a class definition with no content, put in the pass statement to avoid getting an error:  
class Person:  
  pass

**PYTHON INHERITANCE**

Inheritance allows us to define a class that inherits all the methods and properties from another class.  
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.

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()

To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class:

class Student(Person):  
  pass

Use the Student class to create an object, and then execute the printname method:  
x = Student("Mike", "Olsen")  
x.printname()

To keep the inheritance of the parent's \_\_init\_\_() function, add a call to the parent's \_\_init\_\_() function:

class Student(Person):  
  def \_\_init\_\_(self, fname, lname):  
    Person.\_\_init\_\_(self, fname, lname)

Python also has a super() function that will make the child class inherit all the methods and properties from its parent:

class Student(Person):  
  def \_\_init\_\_(self, fname, lname):  
    super().\_\_init\_\_(fname, lname)

**PYTHON SCOPE**

If you need to create a global variable, but are stuck in the local scope, you can use the global keyword.

The global keyword makes the variable global.

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

To change the value of a global variable inside a function, refer to the variable by using the global keyword:

x = 300  
def myfunc():  
  global x  
  x = 200  
myfunc()  
print(x)