**Programming Language PYTHON**

**Notes compiled from various sources**

**Resources:**

Python Website: <https://www.python.org/>

Download Site: <https://www.python.org/download/releases/3.4.1>

Python 3.4.1 was released on May 18th, 2014.

Documentation: <https://docs.python.org/3.4/>

Built-in functions: <https://docs.python.org/3/library/functions.html>

Python Tutorial: <https://docs.python.org/3.4/tutorial/>

Sample Programs: <http://www.cse.msu.edu/~cse231/PracticeOfComputingUsingPython/>

**Python:**

* It is an interpreted language. It can be used interactively (just like Lisp or Prolog).
* It is a ‘typeless’ language. The same variable can assume different types of values at different times.

**Indentation:**

* Python is line oriented, no braces { }. (like Fortran)
* Requires **same amount** of indentation in each block.
* A line can be continued with a backslash character as the last char on the line
* No semicolons at the end of lines

**Identifiers:**

* A-Z, a-z, underscore, 0-9, case-sensitive

**Comments:**

* Begin with # … (as long as # is not in a literal string)

**IDLE editor:** Python distribution comes with an editor that supports Python. Just create a .txt file using notepad and change the name to filename.py. When you right-click on it, one of the options is ‘Edit with IDLE’

**Help function:**

* Provides help on many topics (example: help('print') )

**Assignment Operator**: The equals (=) sign. Multiple assignment allowed.

**>>> a = 5**

**>>> b = 5.6**

**>>> c = "abcde"**

**>>> (d, e, f) = (a+2, b\*2, c+"fgh")**

**>>> d**

**7**

**>>> e**

**11.2**

**>>> f**

**'abcdefgh'**

**Operators:**

**Arithmetic operators: +, -, \*, /, // (integer division), % (mod), \*\* (power)**

**Logical Operators: and, or, not**

**Boolean Constants: True, False**

**Just like in C: zero is False, and non-zero is true**

**Strings and Quotation:** You can use single quotes or double quotes for string literals – as long as you use the same for begin and end of a string. The triple quotes can be used to span the string across multiple lines. For example, all the following are legal:

**word = 'word'**

**sentence = "This is a sentence."**

**paragraph = """This is a paragraph. It is**

**made up of multiple lines and sentences."""**

(\n in the string in a single or double or triple quoted string is interpreted as an end of line)

**Strings are immutable:**  This means that we cannot change a single character in a string. We can assign the whole string.

**x = "abcdef" This is OK.**

**x[2] = 'h' This is not OK**

**x = "abhdef" This is OK.**

**Reserved Words:**

|  |  |  |
| --- | --- | --- |
| **and** | **exec** | **not** |
| **assert** | **finally** | **or** |
| **break** | **for** | **pass** |
| **class** | **from** | **print** |
| **continue** | **global** | **raise** |
| **def** | **if** | **return** |
| **del** | **import** | **try** |
| **elif** | **In** | **while** |
| **else** | **Is** | **with** |
| **except** | **lambda** | **yield** |

**Built-in Functions**

|  |  | **Built-in Functions** |  |  |
| --- | --- | --- | --- | --- |
| [**abs()**](https://docs.python.org/3/library/functions.html#abs) | [**dict()**](https://docs.python.org/3/library/functions.html#func-dict) | [**help()**](https://docs.python.org/3/library/functions.html#help) | [**min()**](https://docs.python.org/3/library/functions.html#min) | [**setattr()**](https://docs.python.org/3/library/functions.html#setattr) |
| [**all()**](https://docs.python.org/3/library/functions.html#all) | [**dir()**](https://docs.python.org/3/library/functions.html#dir) | [**hex()**](https://docs.python.org/3/library/functions.html#hex) | [**next()**](https://docs.python.org/3/library/functions.html#next) | [**slice()**](https://docs.python.org/3/library/functions.html#slice) |
| [**any()**](https://docs.python.org/3/library/functions.html#any) | [**divmod()**](https://docs.python.org/3/library/functions.html#divmod) | [**id()**](https://docs.python.org/3/library/functions.html#id) | [**object()**](https://docs.python.org/3/library/functions.html#object) | [**sorted()**](https://docs.python.org/3/library/functions.html#sorted) |
| [**ascii()**](https://docs.python.org/3/library/functions.html#ascii) | [**enumerate()**](https://docs.python.org/3/library/functions.html#enumerate) | [**input()**](https://docs.python.org/3/library/functions.html#input) | [**oct()**](https://docs.python.org/3/library/functions.html#oct) | [**staticmethod()**](https://docs.python.org/3/library/functions.html#staticmethod) |
| [**bin()**](https://docs.python.org/3/library/functions.html#bin) | [**eval()**](https://docs.python.org/3/library/functions.html#eval) | [**int()**](https://docs.python.org/3/library/functions.html#int) | [**open()**](https://docs.python.org/3/library/functions.html#open) | [**str()**](https://docs.python.org/3/library/functions.html#func-str) |
| [**bool()**](https://docs.python.org/3/library/functions.html#bool) | [**exec()**](https://docs.python.org/3/library/functions.html#exec) | [**isinstance()**](https://docs.python.org/3/library/functions.html#isinstance) | [**ord()**](https://docs.python.org/3/library/functions.html#ord) | [**sum()**](https://docs.python.org/3/library/functions.html#sum) |
| [**bytearray()**](https://docs.python.org/3/library/functions.html#bytearray) | [**filter()**](https://docs.python.org/3/library/functions.html#filter) | [**issubclass()**](https://docs.python.org/3/library/functions.html#issubclass) | [**pow()**](https://docs.python.org/3/library/functions.html#pow) | [**super()**](https://docs.python.org/3/library/functions.html#super) |
| [**bytes()**](https://docs.python.org/3/library/functions.html#bytes) | [**float()**](https://docs.python.org/3/library/functions.html#float) | [**iter()**](https://docs.python.org/3/library/functions.html#iter) | [**print()**](https://docs.python.org/3/library/functions.html#print) | [**tuple()**](https://docs.python.org/3/library/functions.html#func-tuple) |
| [**callable()**](https://docs.python.org/3/library/functions.html#callable) | [**format()**](https://docs.python.org/3/library/functions.html#format) | [**len()**](https://docs.python.org/3/library/functions.html#len) | [**property()**](https://docs.python.org/3/library/functions.html#property) | [**type()**](https://docs.python.org/3/library/functions.html#type) |
| [**chr()**](https://docs.python.org/3/library/functions.html#chr) | [**frozenset()**](https://docs.python.org/3/library/functions.html#func-frozenset) | [**list()**](https://docs.python.org/3/library/functions.html#func-list) | [**range()**](https://docs.python.org/3/library/functions.html#func-range) | [**vars()**](https://docs.python.org/3/library/functions.html#vars) |
| [**classmethod()**](https://docs.python.org/3/library/functions.html#classmethod) | [**getattr()**](https://docs.python.org/3/library/functions.html#getattr) | [**locals()**](https://docs.python.org/3/library/functions.html#locals) | [**repr()**](https://docs.python.org/3/library/functions.html#repr) | [**zip()**](https://docs.python.org/3/library/functions.html#zip) |
| [**compile()**](https://docs.python.org/3/library/functions.html#compile) | [**globals()**](https://docs.python.org/3/library/functions.html#globals) | [**map()**](https://docs.python.org/3/library/functions.html#map) | [**reversed()**](https://docs.python.org/3/library/functions.html#reversed) | [**\_\_import\_\_()**](https://docs.python.org/3/library/functions.html#__import__) |
| [**complex()**](https://docs.python.org/3/library/functions.html#complex) | [**hasattr()**](https://docs.python.org/3/library/functions.html#hasattr) | [**max()**](https://docs.python.org/3/library/functions.html#max) | [**round()**](https://docs.python.org/3/library/functions.html#round) |  |
| [**delattr()**](https://docs.python.org/3/library/functions.html#delattr) | [**hash()**](https://docs.python.org/3/library/functions.html#hash) | [**memoryview()**](https://docs.python.org/3/library/functions.html#func-memoryview) | [**set()**](https://docs.python.org/3/library/functions.html#func-set) |  |

**Input function:**

**input**([*prompt*])[¶](https://docs.python.org/3/library/functions.html#input)

If the *prompt* argument is present, it is written to standard output without a trailing newline. The function then reads a line from input, converts it to a **string** (stripping a trailing newline), and returns that. When EOF is read, [EOFError](https://docs.python.org/3/library/exceptions.html#EOFError) is raised. Example:

**>>> x = input("Enter a number: ")**

**Enter a number: 20**

**>>> x**

**'20'**

**Print function:**

print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.

**Optional** keyword arguments: **file**: a file-like object (stream); defaults to the current sys.stdout.  **sep:** string inserted between values, default a space. **end:**  string appended after the last value, default a newline. **flush:** whether to forcibly flush the stream.

**>>> print("Hello World");**

**Hello World**

**Control Statements (BNF)**

**if Statement: while Statement**

**While <expression>:**

**<suite>**

**[else:**

**<suite>]**

**if <expression>:**

**<suite>**

**(elif <expression>:**

**<suite>)\***

**[else:**

**<suite>]**

**for <target> in <expression\_list>:**

**<suite>**

**[else:**

**<suite>]**

**Examples:** (Observe the conversion from String to int in the first line below)

**>>> x = int(input("Please enter an integer: "))**

**Please enter an integer: 42**

**>>> if x < 0:**

**... x = 0**

**... print('Negative changed to zero')**

**... elif x == 0:**

**... print('Zero')**

**... elif x == 1:**

**... print('Single')**

**... else:**

**... print('More')**

**...**

**More**

**Here we create a list and loop through it to print each element and its length. Observe the built-in string function len.**

**>>> words = ['cat', 'window', 'defenestrate']**

**>>> for w in words:**

**... print(w, len(w))**

**...**

**cat 3**

**window 6**

**defenestrate 12**

**A while loop that prints numbers**

**>>> x = 1000**

**>>> while x > 0:**

**... print(x)**

**... x = x // 2 #observe //, integer division**

**... else:**

**... print("x must be > 0")**

**...**

**1000**

**500**

**250**

**125**

**62**

**31**

**15**

**7**

**3**

**1**

**x must be > 0**

**Range Function:**

* **range(m,n) All integers from m to n-1, default increment 1**
* **range(m,n,k) Integers m, m+k, m+2k, etc. increment = k**

**>>> for x in range(1,10,2):**

**... print(x)**

**...**

**1**

**3**

**5**

**7**

**9**

**>>> for x in range(2, 10):**

**... print(x)**

**...**

**2**

**3**

**4**

**5**

**6**

**7**

**8**

**9**

**>>> a = ['Mary', 'had', 'a', 'little', 'lamb']**

**>>> for i in range(len(a)):**

**... print(i, a[i])**

**...**

**0 Mary**

**1 had**

**2 a**

**3 little**

**4 lamb**

**Prime Numbers:** In this example we examine the numbers 2 to 10 and try to divide each number n by another number in the range 2 to n

**>>> for n in range(2, 10):**

**... for x in range(2, n):**

**... if n % x == 0:**

**... print(n, 'equals', x, '\*', n//x)**

**... break**

**... else:**

**... # loop fell through without finding a factor**

**... print(n, 'is a prime number')**

**...**

**2 is a prime number**

**3 is a prime number**

**4 equals 2 \* 2**

**5 is a prime number**

**6 equals 2 \* 3**

**7 is a prime number**

**8 equals 2 \* 4**

**9 equals 3 \* 3**

**Datatypes: (https://docs.python.org/3/library/)**

[**datetime — Basic date and time types**](https://docs.python.org/3/library/datetime.html)

[**calendar — General calendar-related functions**](https://docs.python.org/3/library/calendar.html)

[**collections — Container datatypes**](https://docs.python.org/3/library/collections.html)

[**collections.abc — Abstract Base Classes for Containers**](https://docs.python.org/3/library/collections.abc.html)

[**heapq — Heap queue algorithm**](https://docs.python.org/3/library/heapq.html)

[**bisect — Array bisection algorithm**](https://docs.python.org/3/library/bisect.html)

[**array — Efficient arrays of numeric values**](https://docs.python.org/3/library/array.html)

[**weakref — Weak references**](https://docs.python.org/3/library/weakref.html)

[**types — Dynamic type creation and names for built-in types**](https://docs.python.org/3/library/types.html)

[**copy — Shallow and deep copy operations**](https://docs.python.org/3/library/copy.html)

[**pprint — Data pretty printer**](https://docs.python.org/3/library/pprint.html)

[**reprlib — Alternate repr() implementation**](https://docs.python.org/3/library/reprlib.html)

[**enum — Support for enumerations**](https://docs.python.org/3/library/enum.html)

**Type Coercion:**

int (anything that can be converted into a int)

**Examples**

**>>> int(5.8)**

**5**

**>>> int("32")**

**32**

**>>> int("hello")**

**Traceback (most recent call last):**

**File "<stdin>", line 1, in <module>**

**ValueError: invalid literal for int() with base 10: 'hello'**

**>>> int("35.6")**

**Traceback (most recent call last):**

**File "<stdin>", line 1, in <module>**

**ValueError: invalid literal for int() with base 10: '35.6'**

**>>> int(35.6)**

**35**

**float(anything that can be converted to float)**

**>>> float(34)**

**34.0**

**>>> float(35+36)**

**71.0**

**>>> float("35")**

**35.0**

**>>> float("45.92")**

**45.92**

**>>> float("25.6"+"35.9")**

**Traceback (most recent call last):**

**File "<stdin>", line 1, in <module>**

**ValueError: could not convert string to float: '25.635.9'**

**>>> float("1.25E+02")**

**125.0**

**str(anything) --**

**Predefined Modules:** There are a lot of predefined modules (<https://docs.python.org/3/library/>)

Examples:

[**numbers — Numeric abstract base classes**](https://docs.python.org/3/library/numbers.html)

[**math — Mathematical functions**](https://docs.python.org/3/library/math.html)

[**cmath — Mathematical functions for complex numbers**](https://docs.python.org/3/library/cmath.html)

[**decimal — Decimal fixed point and floating point arithmetic**](https://docs.python.org/3/library/decimal.html)

[**fractions — Rational numbers**](https://docs.python.org/3/library/fractions.html)

[**random — Generate pseudo-random numbers**](https://docs.python.org/3/library/random.html)

[**statistics — Mathematical statistics functions**](https://docs.python.org/3/library/statistics.html)

**math module:**

**import math**

**math.log(number) -- here base is e**

**math.log10(number) – here base is 10**

**Trig functions: sin, cos, tan,**

**sqrt(number)**

**exp(number) e raised to the power of number**

**pow(base, exp)**

**- The string module**

**>>>import string**

**>>>str = "banana"**

**>>>str.find("a") # there is a built-in find function**

**1**

**>>>str.find("a", 2) # find 'a' starting from index 2 in original string**

**3**

**>>> str.endswith("na")**

**True**

**>>> str.endswith("nana")**

**True**

**>>> str.endswith("nanana")**

**False**

**>>>>>> str.upper() # similarly, you can have 'str.lower()'**

**'BANANA'**

**'+' ALSO WORKS WITH strings IN Python JUST LIKE IN Java (string concatenation)**

**>>> x = "abcde"**

**>>> y = "xyz"**

**>>> x+y**

**'abcdexyz'**

**You may capitalize only the first letter of a string**

**>>> str = str.lower()**

**>>> print(str)**

**banana**

**>>> str.capitalize()**

**'Banana'**

**split() method:** Splits the string into a list of strings based on a separator

**>>> str = "banana"**

**>>> str.split('a') #split using 'a' as the separator**

**['b', 'n', 'n', '']**

**>>> str = "Able was I ere I saw Elba"**

**# We want to split this into a list based on the blank**

**>>> str.split(' ')**

**['Able', 'was', 'I', 'ere', 'I', 'saw', 'Elba']**

**>>>**

**================================================================**

**Functions**

**How to define functions**

**Syntax**

**def <fname> (<list\_of\_params>):**

**<statements>**

**Examples:**

**def printTwice (arg):**

**print (arg, arg)**

**import math**

**def area(radius):**

**temp = math.pi \* radius \*\* 2**

**return temp**

**Example of a recursive function**

**def fact(num):**

**if num < 0:**

**print("Illegal Parameter for fact function")**

**return -1**

**elif num == 0:**

**return 1**

**else:**

**return num \* fact(num-1)**

**Boolean Function Example**

**def isDivisible(x, y): # is x divisible by y?**

**return x%y == 0**

**>>> if isDivisible(100, 5):**

**... print("100 is divisible by 5")**

**... else:**

**... print("100 is NOT divisible by 5")**

**...**

**100 is divisible by 5**

**Local variables:**

Variables don’t need to be declared. The first use is an implicit declaration. Type is dynamic ( k = 1; k = "one"). Scope rules apply.

**Write a function that prints a string backwards one letter per line.**

**def printBackwards(word):**

**n = len(word)**

**while (n > 0):**

**print (word[n-1])**

**n = n-1**

**A 'find()' function for strings:** Find a character in a string. Return the location of the first match or -1 if not found.

**def find(str, ch):**

**index = 0**

**while index < len(str):**

**if str[index] == ch:**

**return index**

**index = index + 1**

**return -1**

**def find2(str, ch, startIndex): # find first occurrence of character 'ch' from 'startIndex'**

**index = startIndex**

**while index < len(str):**

**if str[index] == ch:**

**return index**

**index = index + 1**

**return -1**

**def find3(str, ch):**

**#find3 is same as find, except it uses find2**

**return find2(str, ch, 0)**

**LISTS:** Lists may be NESTED

For example:

**>>> list1 = ["hello", 2.0, 5, [10, 20]]**

**>>> list1**

**['hello', 2.0, 5, [10, 20]]**

**>>> list1[3]**

**[10, 20]**

**>>> list1[3][1]**

**20**

**>>>**

**len() function**: The 'len()' function gets you the length of a list. In fact, len function works on compound types and returns the number of elements.

**'in' operator:** Checks for membership in a list

**>>> 5 in list1**

**True**

**The 'in' operator is useful in a for loop involving lists (also works with strings)**

**>>> for item in list1:**

**... print (item)**

**...**

**hello**

**2.0**

**5**

**[10, 20]**

**Note the use of the '+' operator for string concatenation**

**>>> for fruit in ["apples", "grapes", "oranges"]:**

**... print("I like to eat " + fruit + "!")**

**...**

**I like to eat apples!**

**I like to eat grapes!**

**I like to eat oranges!**

**The '+' operator can also be used for list concatenation**

**>>> list3 = [5, 6, 7] + [8, 9]**

**>>> list3**

**[5, 6, 7, 8, 9]**

**Slicing:** We can creates ‘slices’ of lists (i.e create sub-lists beginning and ending at some points)

**[m:n] means m, m+1, …, n-1**

**words[:n] beginning up to n-1**

**words[m:] from m to end**

**words[:] all of them**

**>>> squares = [1,4,9,16,25,36,49,64,81,100]**

**>>> squares # This returns the whole list**

**[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]**

**>>> squares[0] # works like an array with index beginning at zero**

**1**

**>>> squares[2:5] # slice from index 2 to index 4**

**[9, 16, 25]**

**>>> squares[:7] # slice from index 0 to index 6**

**[1, 4, 9, 16, 25, 36, 49]**

**>>> squares[3:] # slice from index 3 to the end**

**[16, 25, 36, 49, 64, 81, 100]**

**>>> squares[2:20] # if the second index is too large, it just gives till end**

**[9, 16, 25, 36, 49, 64, 81, 100]**

**>>> squares[-1] # It is like wraparound: -1 is the last element**

**100**

**>>> squares[-4] # 4th element from the end**

**49**

**>>> squares[-4:] # slice from 4th element from end up to the end of list**

**[49, 64, 81, 100]**

**Lists are Mutable:** Unlike strings, which are [*immutable*](https://docs.python.org/3.4/glossary.html#term-immutable), lists are a [*mutable*](https://docs.python.org/3.4/glossary.html#term-mutable) type, i.e. it is possible to change their content:

**>>> cubes = [1, 8, 27, 65, 125] # something's wrong here**

**>>> cubes[3] = 64 #Now this is right**

**>>> cubes**

**[1, 8, 27, 64, 125]**

**You can append to a list:** (You can append one element at a time)

**>>> cubes.append(216)**

**>>> cubes**

**[1, 8, 27, 64, 125, 216]**

**>>> cubes.append(7 \*\* 3)**

**>>> cubes**

**[1, 8, 27, 64, 125, 216, 343]**

**Assignment to slices is also possible, and this can even change the size of the list or clear it entirely:**

**>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']**

**>>> letters**

**['a', 'b', 'c', 'd', 'e', 'f', 'g']**

**>>> # replace some values**

**>>> letters[2:5] = ['C', 'D', 'E']**

**>>> letters**

**['a', 'b', 'C', 'D', 'E', 'f', 'g']**

**>>> # now remove them**

**>>> letters[2:5] = []**

**>>> letters**

**['a', 'b', 'f', 'g']**

**>>> # clear the list by replacing all the elements with an empty list**

**>>> letters[:] = []**

**>>> letters**

**[]**

**You can also indicate the steps by which you want to compute the slice.**

**>>> list3**

**[5, 6, 7, 8, 9]**

**>>> list3[1:5:1]**

**[6, 7, 8, 9] # Note this output, and look at the one below**

**>>> list3[1:5:2]**

**[6, 8] # Here we went in steps of 2, so we skipped the 7 and the 9. Result list ended at 8, because after skipping 9,**

**# there is nothing -- the list ends**

**>>>**

**An imaginative way of reversing a list**

**>>> list3[::-1]**

**[9, 8, 7, 6, 5]**

**# you went from the first index (0 - implicitly stated here) to last**

**# index (length of list – implicitly stated here) in steps of -1.**

**# This means you started from last index and went to first index**

**# in steps of 1.**

**>>>**

**Del() function:** del() removes an element from a list

**>>> del list3[1]**

**>>> list3**

**[5, 7, 8, 9]**

**>>> del list3[1:3] # can delete a slice of a list**

**>>> list3**

**[5, 9]**

**Lists are OBJECTs in Python. Every object has a unique id, which can be accessed via the id() function**

**>>> a = [1, 2, 3]**

**>>> b = [1, 2, 3]**

**>>> id(a)**

**36042472**

**>>> id(b)**

**36077616**

Since the id of 'b' is not the same as 'a', you can see that 'b' is a different object although it has the same value as a

**# But if you do**

**>>> b = a**

**>>> id(b)**

**36042472**

Now, 'b' has the same id as 'a' - we have thus ALIASED 'b' to 'a'

**Deep copying a list:** In order to do a DEEP COPY, do cloning of the list using the slice operator

**>>> b = a[:]**

**>>> id(b)**

**36077616 # 'b' has a DIFFERENT id again!**

**>>>**

**Passing list as a parameter:** Lists passed as parameters are passed by REFERENCE. A change made to a list by a function is reflected in the actual parameter

**def deleteHead(list):**

**del list[0]**

**>>> numbers = [1, 2, 3]**

**>>> deleteHead(numbers)**

**>>> numbers**

**[2, 3] # Clearly the results of call by reference - numbers has changed.**

**>>>**

**Scalars are passed by value**

If you wish to avoid changes to your actual parameter list, clone the list (using the slicing operator) in the function

**def tail(list):**

**temp = list[:]**

**return temp[1:]**

**Tuples:** A list is mutable. A tuple is an immutable list. Tuple is enclosed in parentheses not square brackets.

**>>> list1 = [1, 2, 3, 4]**

**>>> tuple1 = (1,2,3,4)**

**>>> type(list1)**

**<class 'list'>**

**>>> type(tuple1)**

**<class 'tuple'>**

**>>> list1[2] = 100**

**>>> list1**

**[1, 2, 100, 4]**

**>>> tuple1[1] = 100**

**Traceback (most recent call last):**

**File "<stdin>", line 1, in <module>**

**TypeError: 'tuple' object does not support item assignment**

**>>> tuple1 = (1, 2, 100, 4)**

**>>> tuple1**

**(1, 2, 100, 4)**

**You can take slices of tuples. Even if there is only one element the tuple has to have a comma. (See below)**

**>>> tuple1[2:3]**

**(100,)**

**Easy to swap two variables using tuple assignment:**

When a multiple assignment such as a,b,c,d = 1,2,3,4 is executed, the system actually makes a tuple of addresses of a, b, c and d and assigns them from a tuple of values from the right-hand-side. This gives us a convenient way of swapping two elements. For example: a,b = b,a swaps them.

**Tuples are first class objects and functions can return them.**

**Here is swap function:**

**def swap(a, b):**

**return b,a**

**>>> swap(c,d)**

**(4, 3)**

**>>> c,d**

**(3, 4)**

**>>> c,d = swap(c,d)**

**>>> c,d**

**(4, 3)**

**Dictionaries:** A dictionary is a collection of (key, value) pairs, just like a hashtable in Java. The ‘key’ must be an immutable type (such as a string)

**capitals = {} #This is an empty dictionary**

**capitals['New York'] = 'Brockport'**

**capitals['Massachusetts'] = 'Boston'**

**capitals['New Jersey'] = 'Trenton'**

**capitals['Pennsylvania'] = 'Harrisburg'**

**>>> capitals**

**{'New Jersey': 'Trenton', 'Massachusetts': 'Boston', 'Pennsylvania': 'Harrisburg', 'New York': 'Brockport'}**

**>>> capitals["Pennsylvania"]**

**'Harrisburg'**

**>>> capitals["California"]**

**Traceback (most recent call last):**

**File "<stdin>", line 1, in <module>**

**KeyError: 'California'**

**Dictionaries are mutable:**

**capitals ["New York"] = "Albany"**

**You can add to a dictionary:**

**capitals ["California"] = "Sacramento"**

**You can delete from a dictionary:**

**del capitals ["New Jersey"] #This will delete the dictionary entry for New Jersey**

**>>> capitals ["New York"] = "Albany"**

**>>> capitals ["California"] = "Sacramento"**

**>>> capitals**

**{'New Jersey': 'Trenton', 'Massachusetts': 'Boston', 'Pennsylvania': 'Harrisburg', 'New York': 'Albany', 'California': 'Sacramento'}**

**>>> del capitals ["New Jersey"]**

**>>> capitals**

**{'Massachusetts': 'Boston', 'Pennsylvania': 'Harrisburg', 'New York': 'Albany', 'California': 'Sacramento'}**

**Len function for dictionaries returns the number of key-value pairs.**

**Other dictionary Methods:**

**keys() – returns a dict\_keys object of all keys in the dictionary**

**values() – returns dict\_values object.**

**items() – returns dict\_items objects**

**Dictionaries are not lists. But we can convert them into lists:**

**keyList = list(capitals.keys())**

**valueList = list(capitals.values())**

**itemList = list(capitals.items())**

**>>> keyList**

**['Massachusetts', 'Pennsylvania', 'New York', 'California']**

**>>> valueList**

**['Boston', 'Harrisburg', 'Albany', 'Sacramento']**

**>>> itemList**

**[('Massachusetts', 'Boston'), ('Pennsylvania', 'Harrisburg'), ('New York', 'Albany'), ('California', 'Sacramento')]**

**Use the ‘in’ operator to check if a key or a value is present in the dictionary:**

**>>> "California" in capitals**

**True**

**>>> "Texas" in capitals**

**False**

**>>> "California" in capitals.keys()**

**True**

**>>> "Sacramento" in capitals.values()**

**True**

**>>> "Austin" in capitals.values()**

**False**

**>>> "New Jersey" in capitals**

**False**

**>>> "New Jersey" in capitals.values()**

**False**

**>>> "Albany" in capitals**

**False**

**>>> capitals**

**{'Massachusetts': 'Boston', 'Pennsylvania': 'Harrisburg', 'New York': 'Albany', 'California': 'Sacramento'}**

**>>> "Albany" in capitals.values()**

**True**

**Copying dictionaries:**

**dictionary1 = dictionary2 # This is a shallow copy**

**dictionary1 = dictionary2.copy() # This is a deep copy**

**>>> capitals2 = capitals**

**>>> id(capitals)**

**4957808**

**>>> id(capitals2)**

**4957808**

**>>> capitals3 = capitals.copy()**

**>>> id(capitals3)**

**4957768**

**Classes and Objects**

You can create a class with absolutely nothing in it, and even instantiate it:

**class Test: #This is a class with nothing in it**

**pass**

**useless = Test() #create an object of the Test class**

You can add instance variables to it dynamically:

**>>> useless.x = 5.0**

**>>> print(useless.x)**

**5.0**

**>>> useless.y = "abcd"**

**>>> print(useless.y)**

**abcd**

**Importing files (classes in other files into a Python environment or a file)**

Suppose we have two classes called Account and Bank defined in a file called BankingModel.py. We can import this into an active Python environment by typing:

**from BankingModel import Account, Bank or**

**from BankingModel import \***

Such statements can also be in files. For example, in a file called TestBankingModel.py we can have the similar statements.

With this import, you can instantiate a Bank object using: b = Bank() (as if it was defined in the same file)

**How to deep copy an object:** While data structures can be deep-copied using the copy method (list2 = list1.copy()), objects cannot be copies that way. We need to import the copy module

**import copy**

**useless2 = copy.copy(useless)**

**useless3 = useless.copy() #This will give a syntax error**

**id(useless)**

**id(useless2)**

**id(useless3)**

**>>> import copy**

**>>> useless2 = copy.copy(useless)**

**>>> useless3 = useless.copy()**

**Traceback (most recent call last):**

**File "<stdin>", line 1, in <module>**

**AttributeError: 'Test' object has no attribute 'copy'**

**>>> id(useless)**

**22093424**

**>>> id(useless2)**

**22093680**

**>>> id(useless3)**

**Traceback (most recent call last):**

**File "<stdin>", line 1, in <module>**

**NameError: name 'useless3' is not defined**

Copy method (copy.copy) copies only the highest level values. If the object contains other objects encapsulated in it, they are not deep copied. See example below:

**>>> class Point: #define two empty classes**

**... pass**

**...**

**>>> class Rectangle:**

**... pass**

**...**

**>>> r1 = Rectangle() #Make one Rectangle and two Point objects**

**>>> p1 = Point()**

**>>> p2 = Point()**

**>>> p1.x = 5 #assign some random values**

**>>> p1.y = 10**

**>>> p2.x = 20**

**>>> p2.y = 40**

**>>> r1.point1 = p1**

**>>> r1.point2 = p2**

**>>> r2 = copy.copy(r1) #This is an ordinary copy of r1**

**>>> id(r1)**

**22126768**

**>>> id(r2)**

**22364976 # See that id’s of r1 and r2 are different**

**>>> id(r1.point1) # But, the id’s of r1.point1 and r2.point1 are same**

**22127504**

**>>> id(r2.point1)**

**22127504**

**>>> id(r1.point2) # But, the id’s of r1.point2 and r2.point2 are same**

**22127696**

**>>> id(r2.point2)**

**22127696**

**>>> r3 = copy.deepcopy(r1) #This is a real deep copy of r1**

**>>> id(r1) # r1 and r3 have different ids**

**22126768**

**>>> id(r3)**

**22364880**

**>>> id(r1.point1) # r1.point1, r3.point1 also have different ids**

**22127504**

**>>> id(r3.point1)**

**22364848**

**>>> id(r1.point2) # r1.point2, r3.point2 also have different ids**

**22127696**

**>>> id(r3.point2)**

**22365104**

**>>>**

**Objects are mutable:** If an object is passed as a parameter to a method, and the method changes the contents of the object – the change persists.

**Classes with Methods in them**

* By convention, the first parameter to a method should be ***self:*** (same as to **this** in Java)
* When a method is called self **should not be** passed as an argument. (Only parameters other than self are passed)

**class <classname>:**

**def <methodname>(<parameters>):**

**<statements>**

**<blankline>**

**<more methods like this>**

**Example: (No blank lines between methods)**

**class Time:**

**def after(self, t2):**

**if self.hour > t2.hour:**

**return 1**

**if self.hour < t2.hour:**

**return 0**

**if self.minute > t2.minute:**

**return 1**

**if self.minute < t2.minute:**

**return 0**

**if self.second > t2.second:**

**return 1**

**return 0**

**def same(self, t2):**

**return (self.hour == t2.hour) and (self.minute == t2.minute) and (self.second == t2.second)**

**def before(self, t2):**

**return (not self.same(t2)) and (not self.after(t2))**

**doneTime = Time()**

**doneTime.hour = 3**

**doneTime.minute = 51**

**doneTime.second = 25**

**currentTime = Time()**

**currentTime.hour = 3**

**currentTime.minute = 30**

**currentTime.second = 15**

**currentTime.same(doneTime)**

**Parameters** can be given default values at the time of function definition. If the parameter id not passed then default will be used.

We can define a find method as follows:

**def find(str, ch, startIndex = 0): # find first occurrence of character 'ch' from 'startIndex'**

**index = startIndex**

**while index < len(str):**

**if str[index] == ch:**

**return index**

**index = index + 1**

**return -1**

When this find method is used with only two parameters, the third one will be assumed to be 0

**Constructors:** The constructor name is **\_\_init\_\_(self, <otherparameters>)**

(Two underscores, init, two underscores) It does not appear that there is any public, protected or private attributes. They are all public by default.

**class Time:**

**def \_\_init\_\_(self, hour=0, minute=0, second=0):**

**self.hour = hour**

**self.minute = minute**

**self.second = second**

**def after(self, t2):**

**if self.hour > t2.hour:**

**return 1**

**if self.hour < t2.hour:**

**return 0**

**if self.minute > t2.minute:**

**return 1**

**if self.minute < t2.minute:**

**return 0**

**if self.second > t2.second:**

**return 1**

**return 0**

**def same(self, t2):**

**return (self.hour == t2.hour) and (self.minute == t2.minute) and (self.second == t2.second)**

**def before(self, t2):**

**return (not self.same(t2)) and (not self.after(t2))**

**def \_\_str\_\_(self): #This is like the toString method of Java**

**return str(self.hour)+":"+str(self.minute)+":"+str(self.second)**

**>>> c = Time(4,30,45)**

**>>> d = Time(5,20,10)**

**>>> c.after(d)**

**0**

**>>> c.before(d)**

**True**

**>>> print(c)**

**4:30:45**

**>>> print(d)**

**5:20:10**

**Inheritance:**

Inheritance denoted by childClass(parentClass). Automatically inherits methods and instance variables

**class Hand(Deck): #This inherits from Deck**

**def \_\_init\_\_(self, name=""):**

**self.cards = []**

**self.name = name**

**def addCard(self, card):**

**self.cards.append(card)**

**def removeCard(self, card):**

**if card in self.cards:**

**self.cards.remove(card)**

**return 1**

**else:**

**return 0**

**>>> myHand.addCard(Card(0,13))**

**>>> print(myHand)**

**Jack of Diamonds**

**Queen of Diamonds**

**King of Clubs**

**>>> myHand.removeCard(Card(1,12))**

**1**

**>>> print(myHand)**

**Jack of Diamonds**

**King of Clubs**

**Attributes may be inherited as long as they have been declared outside the methods**

**class Deck:**

**names = ["a","b","c"]**

**size = 0**

**def \_\_init\_\_(self):**

**self.cards = []**

**for suit in range(0, 4):**

**for rank in range(1, 14):**

**self.cards.append(Card(suit, rank))**

**self.size = len(self.cards)**

**def \_\_str\_\_(self):**

**s = ""**

**for i in range(len(self.cards)):**

**s = s + " "\*i + str(self.cards[i]) + "\n"**

**return s**

**d = Deck()**

**print(d)**

**class Hand(Deck): #This inherits from Deck**

**def \_\_init\_\_(self, name=""):**

**self.cards = []**

**self.name = name**

**def addCard(self, card):**

**self.cards.append(card)**

**def removeCard(self, card):**

**if card in self.cards:**

**self.cards.remove(card)**

**return 1**

**else:**

**return 0**

**>>> h = Hand()**

**>>> h.names**

**['a', 'b', 'c']**

**>>> h.size**

**0**