Introduction to Python

Agenda for Today's Session

- Session 1
  - Introduction to Python
    - Background of Python
    - Installation & running Python
    - Hello world with Python
    - Importance of Indentation
    - Working with IDE's
  - Working with basic syntax of Python
    - Variables and Data types
    - Operators and expressions
    - the control flow
    - Looping
    - Data Structures
    - Strings

What is Python

- Python is a high-level programming language which is:
  - Interpreted: Python is processed at runtime by the interpreter.
  - **Interactive:** You can use a Python prompt and interact with the interpreter directly to write your programs.
  - **Object-Oriented:** Python supports Object-Oriented technique of programming.
  - **Beginner's Language:** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications.

Designed : **Guido van Rossum**

Releases

- Release dates for the major and minor versions:
  - Python 1.0 - January 1994
    - Python 1.5 - December 31, 1997
    - Python 1.6 - September 5, 2000
  - Python 2.0 - October 16, 2000
    - Python 2.1 - April 17, 2001
    - Python 2.2 - December 21, 2001
    - Python 2.3 - July 29, 2003
    - Python 2.4 - November 30, 2004
    - Python 2.5 - September 19, 2006
    - Python 2.6 - October 1, 2008
    - **Python 2.7 - July 3, 2010**
  - **Python 3.0 - December 3, 2008**
    - Python 3.1 - June 27, 2009
    - Python 3.2 - February 20, 2011
    - Python 3.3 - September 29, 2012
    - Python 3.4 - March 16, 2014
    - **Python 3.5 - September 13, 2015**

Features of Python

- Easy to learn, easy to read and easy to maintain.
- **Portable:** It can run on various hardware platforms and has the same interface on all platforms.
- **Extendable:** You can add low-level modules to the Python interpreter.
- **Scalable:** Python provides a good structure and support for large programs.
- Python has support for an **interactive mode** of testing and debugging.
- Python has a broad standard **library** cross-platform.
- Everything in Python is an **object:** variables, functions, even code. Every object has an ID, a type, and a value.

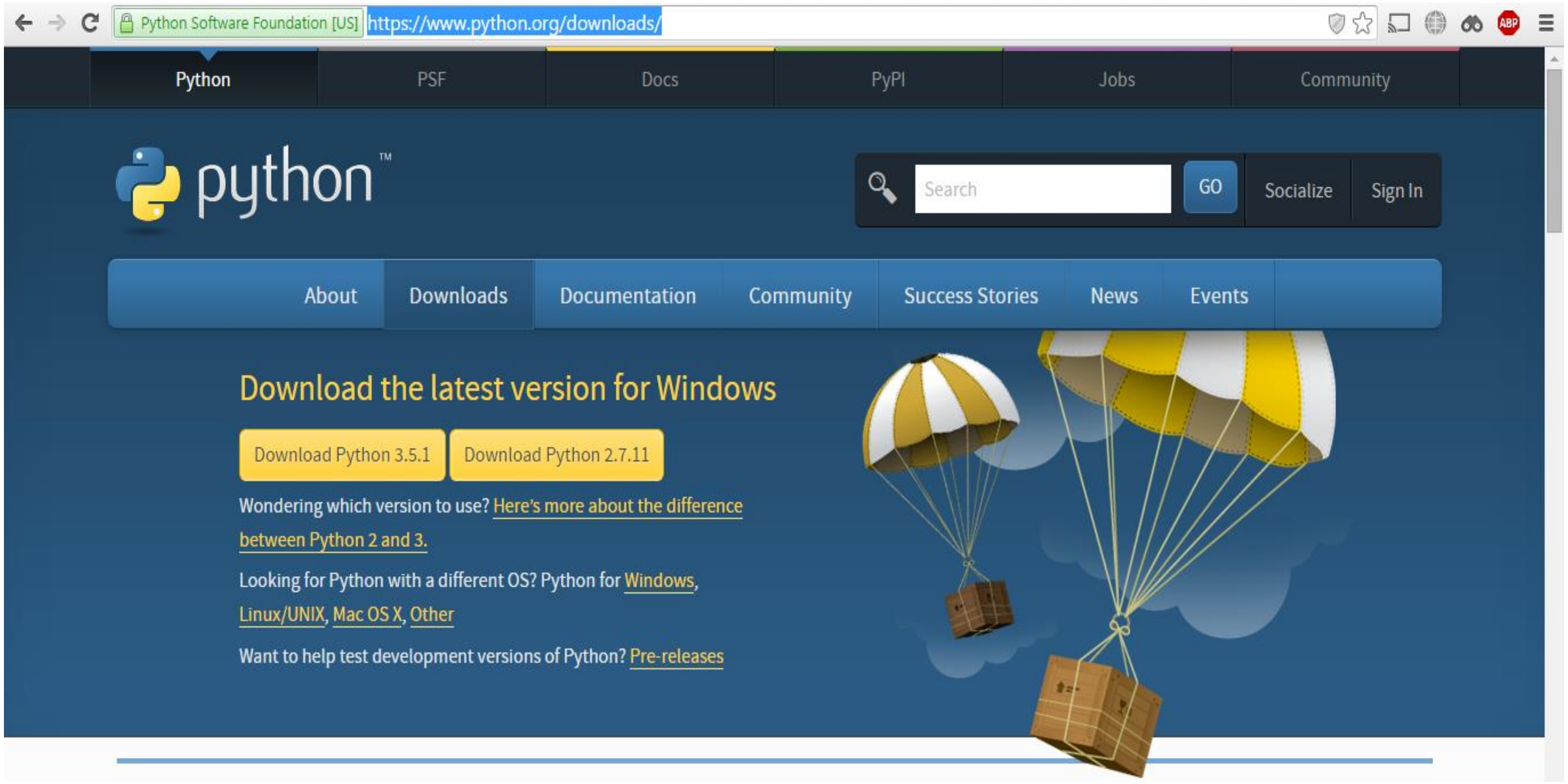
More Features ..

- Python provides interfaces to all major commercial **databases**.
- Python supports functional and structured programming methods as well as **OOP**.
- Python provides very high-level **dynamic** data types and supports dynamic type checking.
- Python supports **GUI** applications
- Python supports automatic **garbage collection.**
- Python can be easily **integrated** with C, C++, and Java.

Who uses Python

grove
cars.com
Confidence Comes Standard.
MIT Massachusetts Institute of Technology
FreeBSD
NetBSD
NASA
Walt Disney Animation Studios
NOKIA Connecting People
YAHOO!
redhat
instagram
evite.com
Atlassian bitbucket
YouTube
National Weather Service
CANONICAL
LexisNexis
Consumer Notebook
Los Angeles Times
EVE ONLINE
ableton
ebay
mozilla Firefox
Google
bit.ly
AstraZeneca
reddit
Quora
TRUECar Know the Real Price
Dropbox
INDUSTRIAL LIGHT & MAGIC

Download Python 3.5.*



Download the latest version for Windows

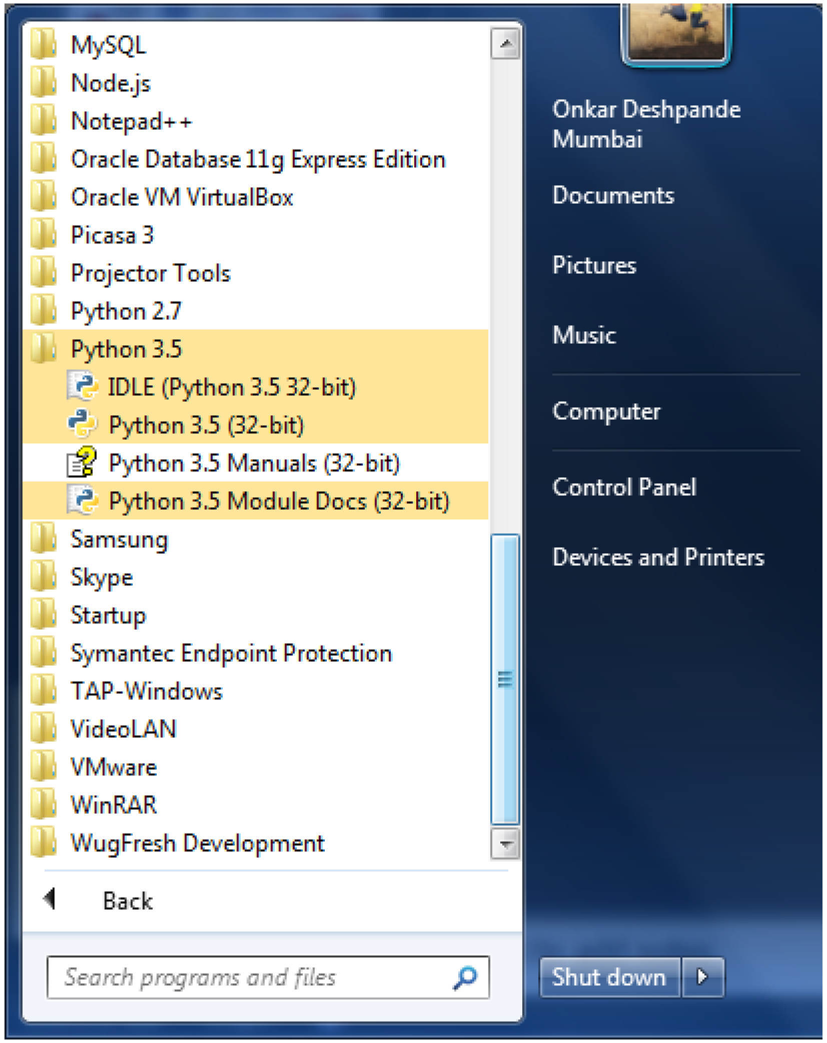
Download Python 3.5.1 | Download Python 2.7.11

Wondering which version to use? Here's more about the difference between Python 2 and 3.

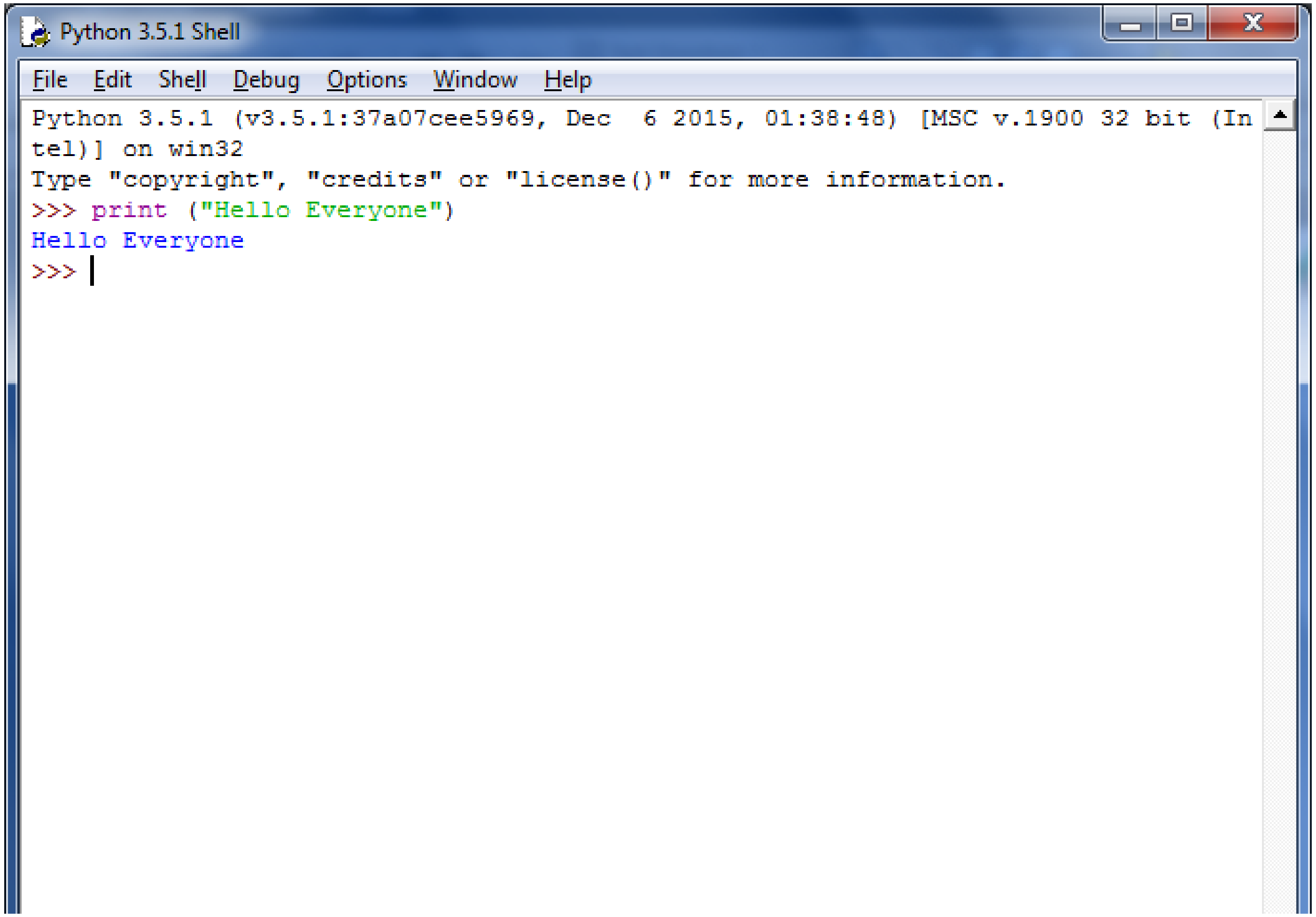
Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases

Run Python Shell

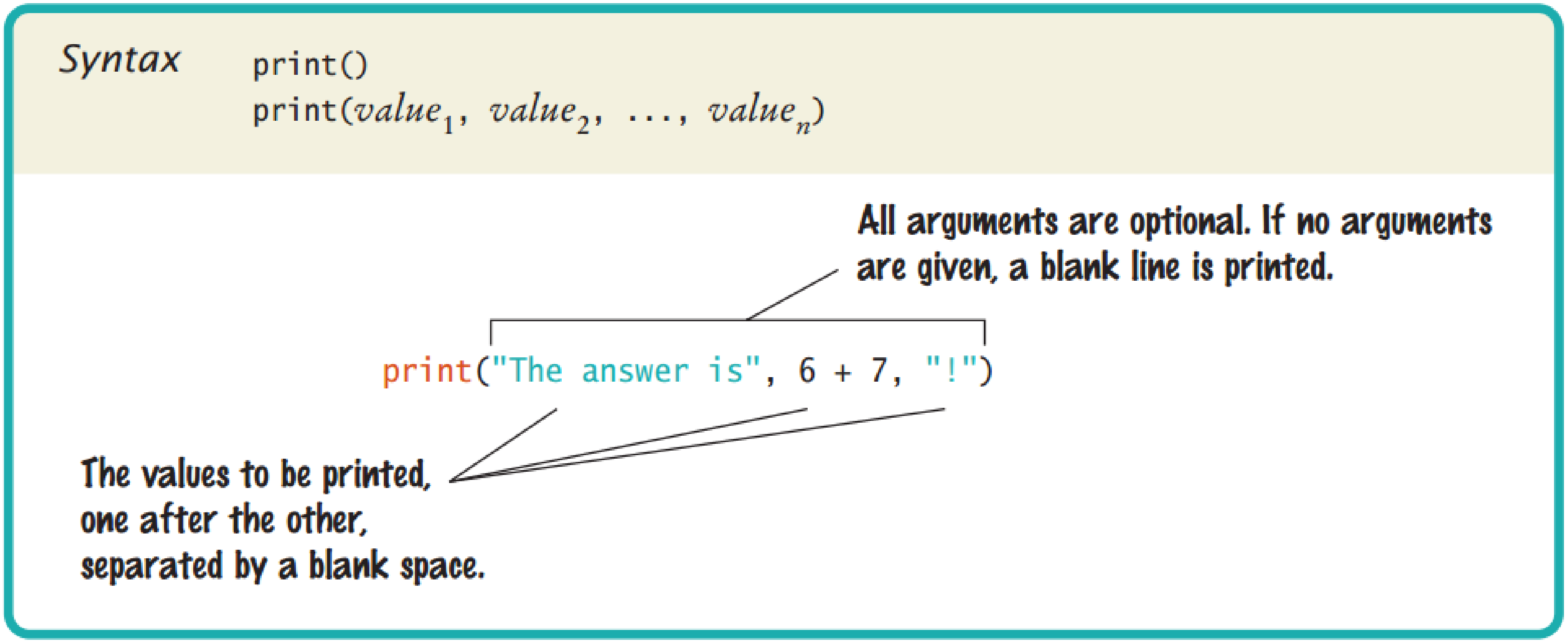


Hello World



```
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  6 2015, 01:38:48) [MSC v.1900 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print ("Hello Everyone")
Hello Everyone
>>> 
```

Print Function Syntax



Syntax

```
print()
print(value_1, value_2, ..., value_n)
```

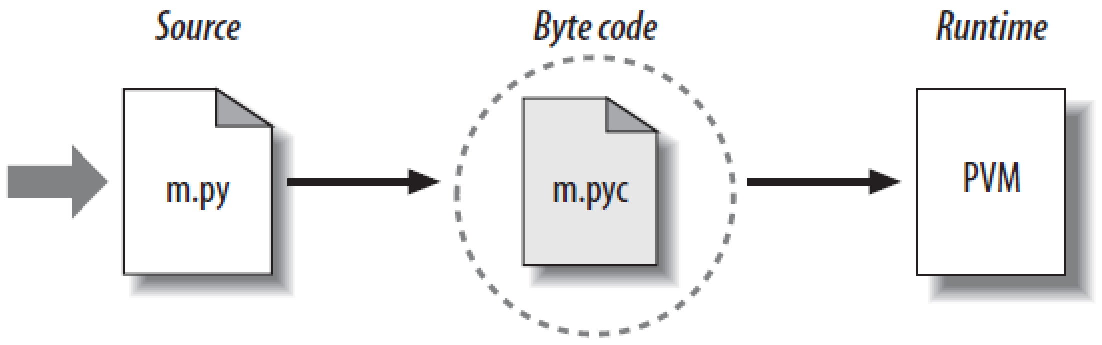
```
print("The answer is", 6 + 7, "!")
```

All arguments are optional. If no arguments are given, a blank line is printed.

The values to be printed, one after the other, separated by a blank space.

Python Code Execution

- Python's traditional runtime execution model: source code you type is translated to byte code, which is then run by the Python Virtual Machine. Your code is automatically compiled, but then it is interpreted.

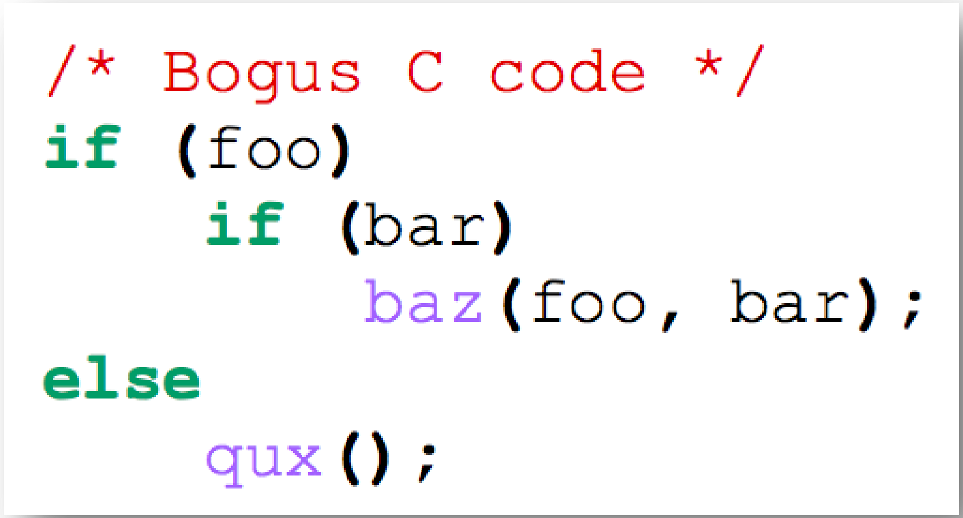


Source code extension is **.py**
Byte code extension is **.pyc** (compiled python code)

Indentation

- Most languages don’t care about indentation
- Most humans do
- We tend to group similar things together

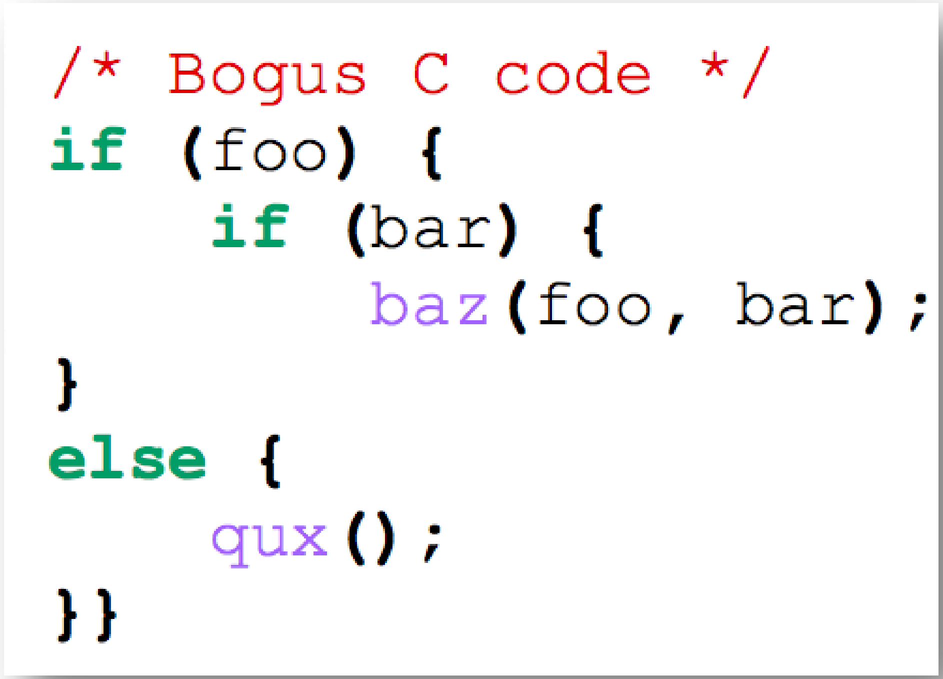
Indentation



```c
/* Bogus C code */
if (foo)
    if (bar)
        baz(foo, bar);
else
    qux();
```

The else here actually belongs to the 2nd if statement

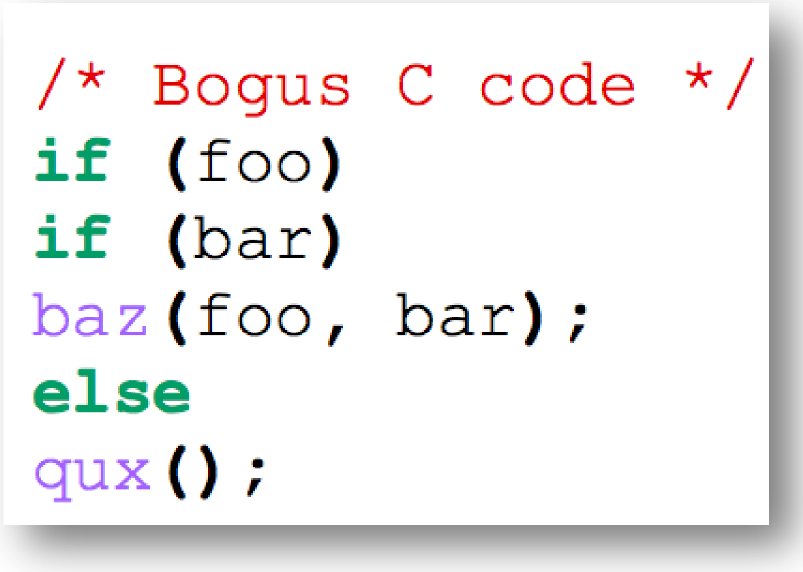
Indentation



```
/* Bogus C code */
if (foo) {
    if (bar) {
        baz(foo, bar);
}
else {
    qux();
}}
```

The else here actually belongs to the 2nd if statement

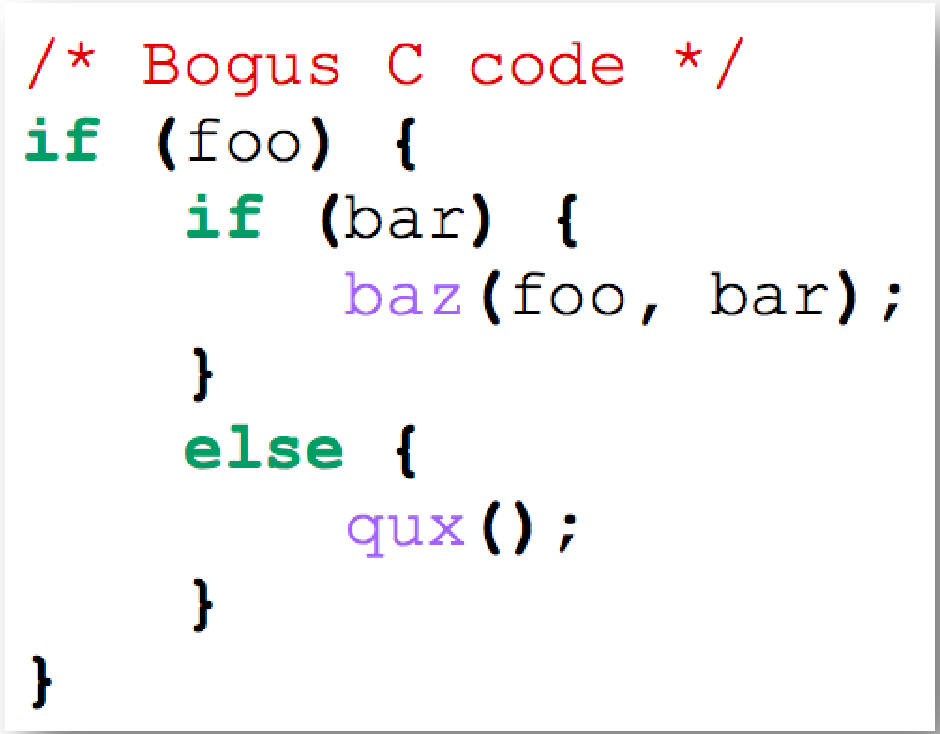
Indentation



```
/* Bogus C code */
if (foo)
if (bar)
baz(foo, bar);
else
qux();
```

I knew a coder like this

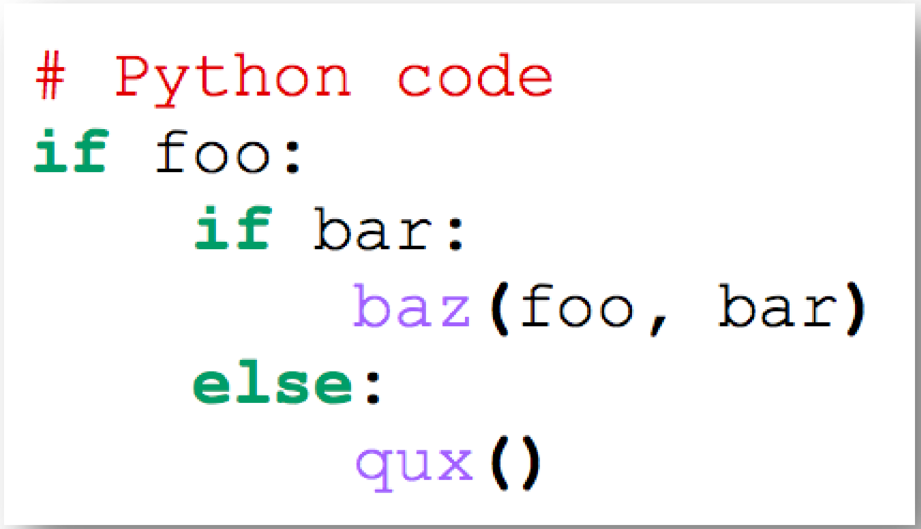
Indentation



```
/* Bogus C code */
if (foo) {
        if (bar) {
                baz(foo, bar);
        }
        else {
                qux();
        }
}
```

You should always be explicit

Indentation



```
# Python code
if foo:
    if bar:
        baz(foo, bar)
    else:
        qux()
```

Python works with indentation closely

Comments

```
# A traditional one line comment

"""
Any string not assigned to a variable is
considered a comment.
This is an example of a multi-line comment.
"""

"This is a single line comment"
```

Working with basic syntax of Python

Variables and Data types

Variables

- In Python we don’t specify what kind of data we are going to put in a variable.

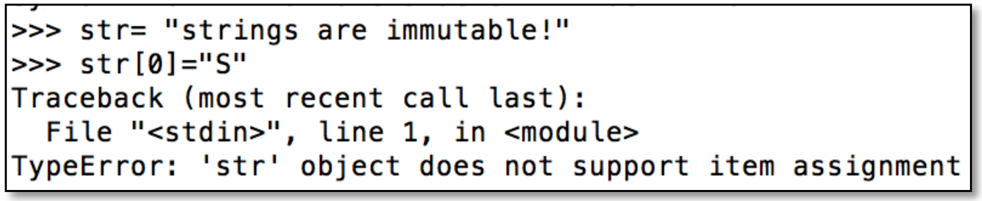
```
a= 45
name= “Onkar”
print (a)
print (name)
```

You can even assign values to multiple variables in a single line. Example

```
a, b = 45, 54
print (a)
print (b)
```

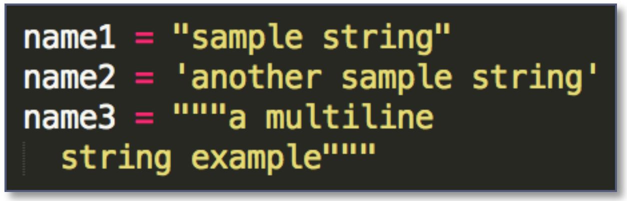
Strings

- Python Strings are Immutable objects that cannot change their values.



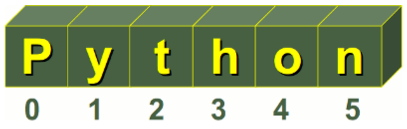
```
>>> str= "strings are immutable!"
>>> str[0]="S"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

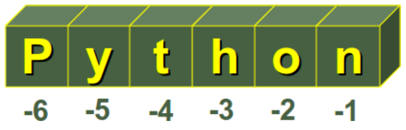
- You can update an existing string by (re)assigning a variable to another string.
- Python does not support a character type; these are treated as strings of length one.
- Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals.



```
name1 = "sample string"
name2 = 'another sample string'
name3 = """a multiline
  string example"""
```

- String indexes starting at 0 in the beginning of the string and working their way from -1 at the end.





Strings

- Common String Methods

Method	Description
str.**count**(sub, beg= 0,end=len(str))	Counts how many times sub occurs in string or in a substring of string if starting index beg and ending index end are given.
str.**isalpha**()	Returns True if string has at least 1 character and all characters are alphanumeric and False otherwise.
str.**isdigit**()	Returns True if string contains only digits and False otherwise.
str.**lower**()	Converts all uppercase letters in string to lowercase.
str.**upper**()	Converts lowercase letters in string to uppercase.
str.**replace**(old, new)	Replaces all occurrences of old in string with new.
str.**split**(str=‘ ’)	Splits string according to delimiter str (space if not provided) and returns list of substrings.
str.**strip**()	Removes all leading and trailing whitespace of string.
str.**title**()	Returns "titlecased" version of string.

- Common String Functions

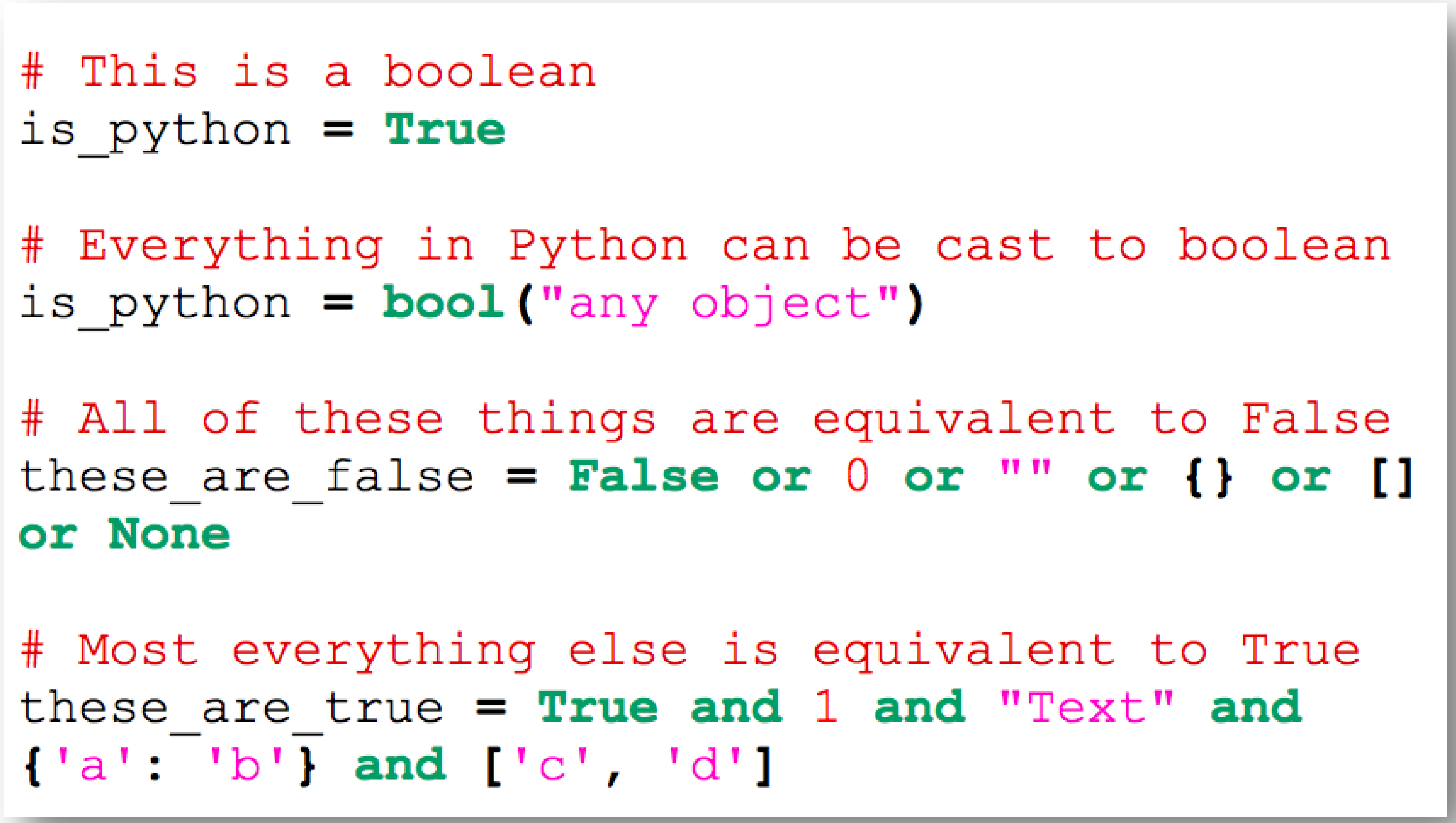
str(x) :to convert x to a string
len(string):gives the total length of the string

Numbers

- Numbers are **Immutable** objects in Python that cannot change their values.
- There are two built-in data types for numbers in Python3:
  - Integer (int)
  - Floating-point numbers (float)
- Common Number Functions

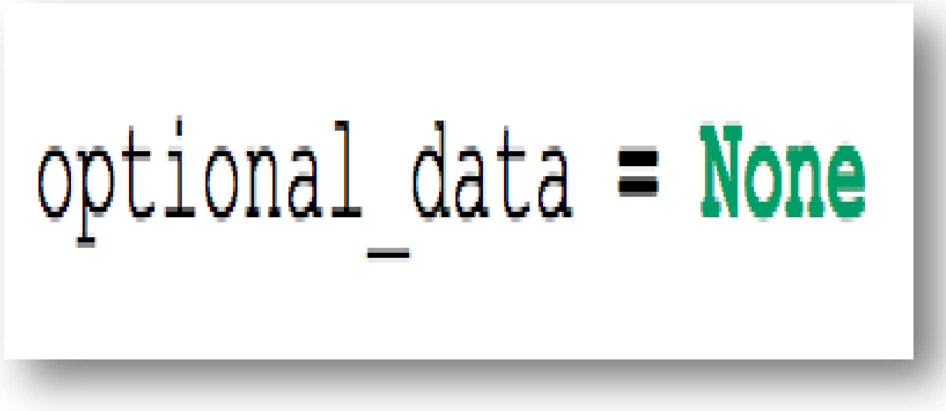
Function	Description
int(x)	to convert x to an integer
float(x)	to convert x to a floating-point number
abs(x)	The absolute value of x
cmp(x,y)	-1 if x < y, 0 if x == y, or 1 if x > y
exp(x)	The exponential of x: e^x
log(x)	The natural logarithm of x, for x> 0
pow(x,y)	The value of x**y
sqrt(x)	The square root of x for x > 0

Booleans



```
# This is a boolean
is_python = True

# Everything in Python can be cast to boolean
is_python = bool("any object")

# All of these things are equivalent to False
these_are_false = False or 0 or "" or {} or []
or None

# Most everything else is equivalent to True
these_are_true = True and 1 and "Text" and
{'a': 'b'} and ['c', 'd']
```

Null



```
optional_data = None
```

Lists

```
#List Creation
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"];

#Accessing Values in Lists
print (list1)
print ("list1[0]: ", list1[0])
print ("list2[1:5]: ", list2[1:3])

#Updating Lists
print ("Value available at index 2 : “)
print (list1[2])
list1[2] = 2001;
print ("New value available at index 2 : “)
print (list1[2])

#Delete List Elements
print (list1)
del list1[2]
print ("After deleting value at index 2 : “ )
print (list1)
```

Built-in List Functions & Methods

- Function with Description
  - cmp(list1, list2) #Compares elements of both lists.
  - len(list) #Gives the total length of the list.
  - max(list) # Returns item from the list with max value.
  - min(list) # Returns item from the list with min value.
  - list(seq) # Converts a tuple into list.

- Methods with Description
  - list.append(obj) # Appends object obj to list
  - list.count(obj) #Returns count of how many times obj occurs in list
  - list.extend(seq) #Appends the contents of seq to list
  - Many more …

Tuples

- Tuples are data separated by commas.

```
a = 'Fedora', 'Debian', 'Kubuntu', 'Pardus'
print a
#('Fedora', 'Debian', 'Kubuntu', 'Pardus')

print a[1]
# 'Debian'

for x in a:
    print(x)
#Fedora Debian Kubuntu Pardus
```

- Tuples are immutable, meaning that you can not del/add/edit any value inside the tuple.

```
#Tuples are immutable
del a[0]
```

Sets

- Sets are another type of data structure with no duplicate items. We can apply mathematical set operations on sets.

```
a = set('abracadabra')
b = set('alacazam')

print a # unique letters in a
#{'a', 'r', 'b', 'c', 'd'}

print a - b # letters in a but not in b
#{'r', 'd', 'b'}

print a | b # letters in either a or b
#{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}

print a & b # letters in both a and b
#{'a', 'c'}

print a ^ b # letters in a or b but not both
#{'r', 'd', 'b', 'm', 'z', 'l'}
```

Dictionaries

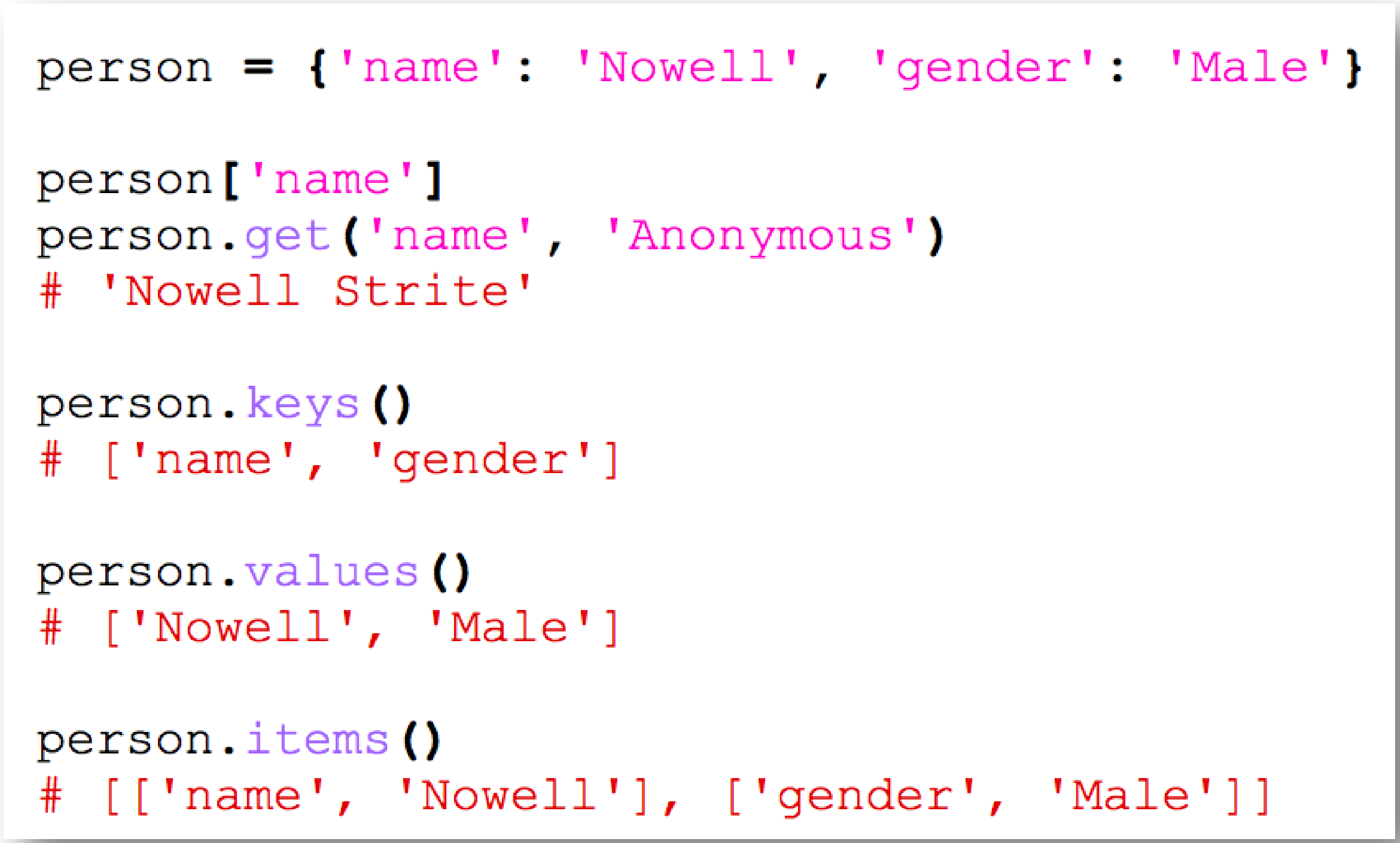
```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};

print ("dict['Name']: ", dict['Name'])
print ("dict['Age']: ", dict['Age'])

#Updating Dictionary

dict['Age'] = 8; # update existing entry
dict['School'] = "DPS School"; # Add new entry


print ("dict['Age']: ", dict['Age'] )
print ("dict['School']: ", dict['School'] )

#Delete Dictionary Elements
del dict['Name']; # remove entry with key 'Name'
dict.clear();     # remove all entries in dict
del dict ;        # delete entire dictionary

print ("dict['Age']: ", dict['Age'] )
print ("dict['School']: ", dict['School'])
```

Built-in Dictionary Methods & functions

- Function with Description
  - cmp(list1, list2) #Compares elements of both lists.
  - len(list) #Gives the total length of the list.
  - str(dict) #Produces a printable string representation of a dictionary
  - type(variable) # Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.

- Methods with Description
  - dict.clear() # Removes all elements of dictionary *dict*
  - dict.copy() #Returns a shallow copy of dictionary *dict*
  - dict.fromkeys() #Create a new dictionary with keys from seq and values set to value.
  - dict.get(key, default=None) #For *key* key, returns value or default if key not in dictionary
  - Many more …

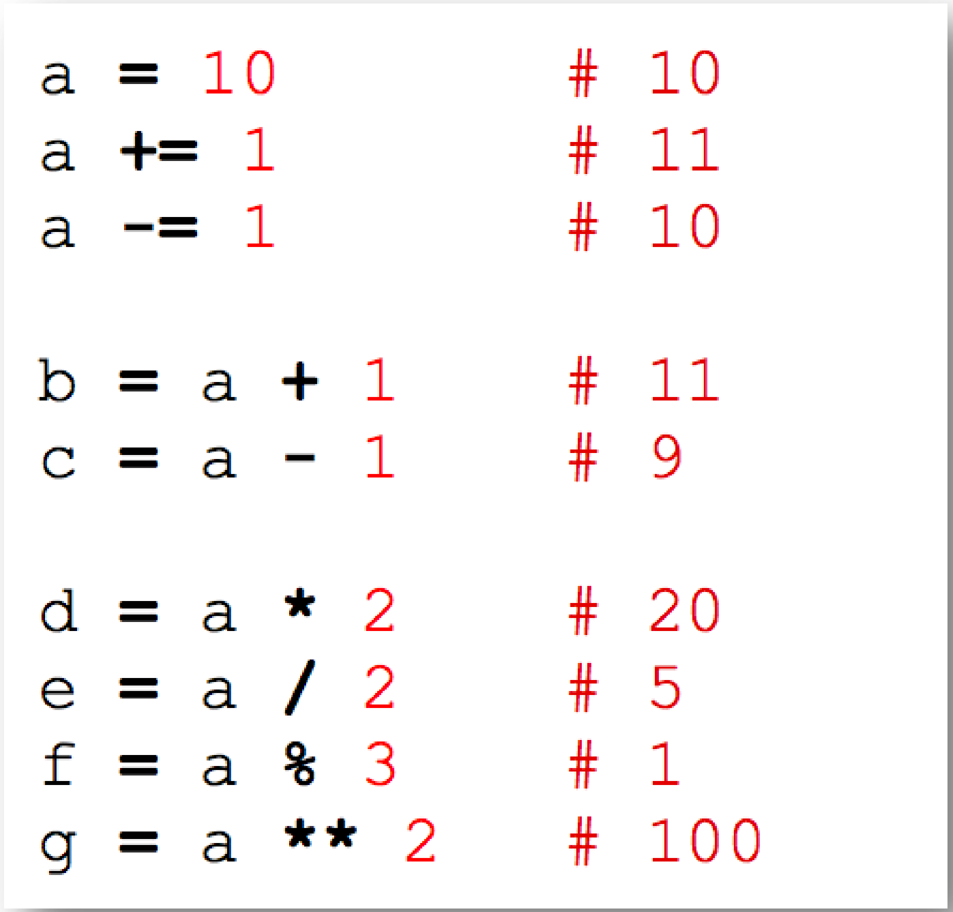
Dictionary Methods



```
person = {'name': 'Nowell', 'gender': 'Male'}

person['name']
person.get('name', 'Anonymous')
# 'Nowell Strite'

person.keys()
# ['name', 'gender']

person.values()
# ['Nowell', 'Male']

person.items()
# [['name', 'Nowell'], ['gender', 'Male']]
```

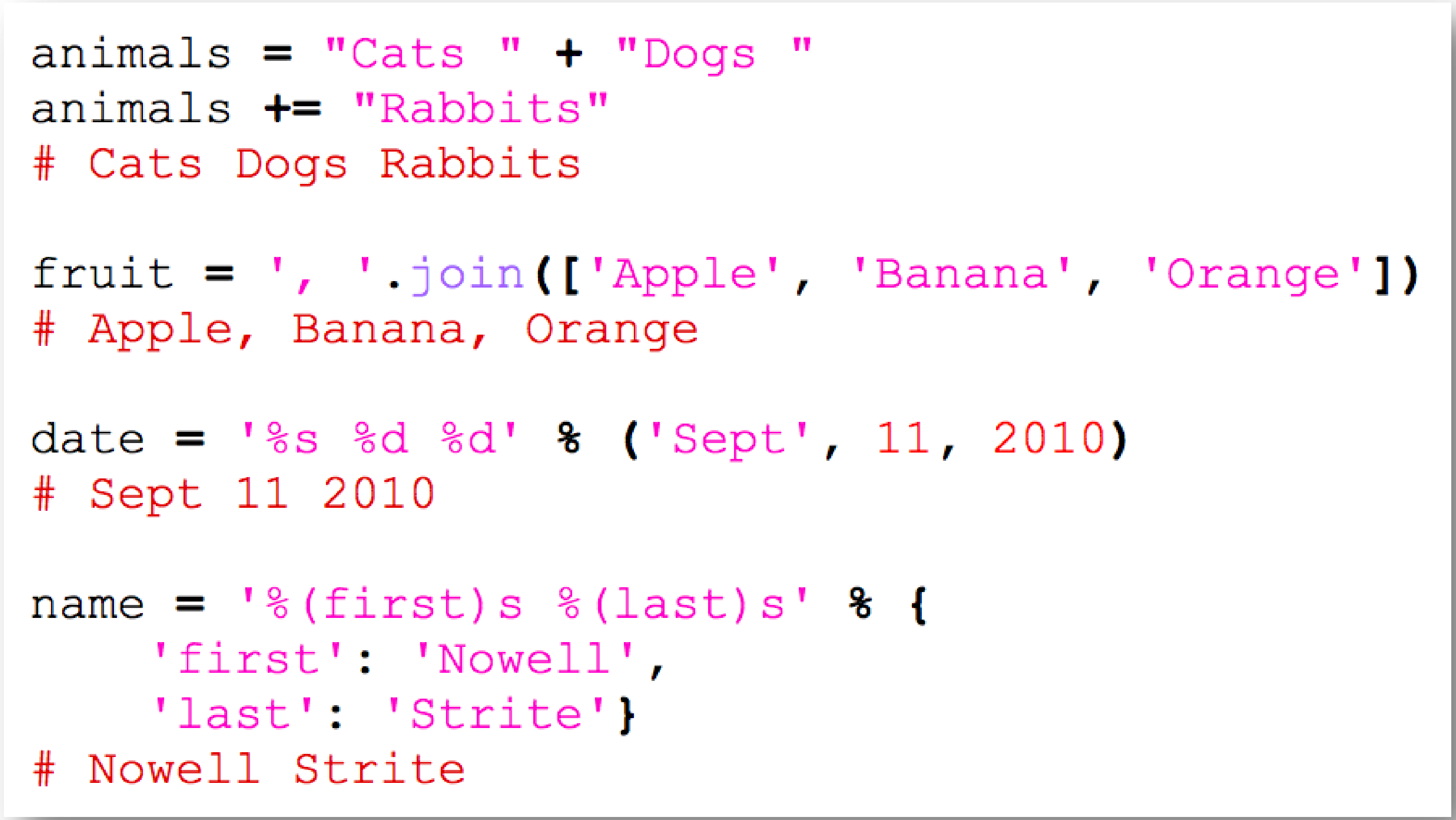
Operators & Expression

Arithmetic



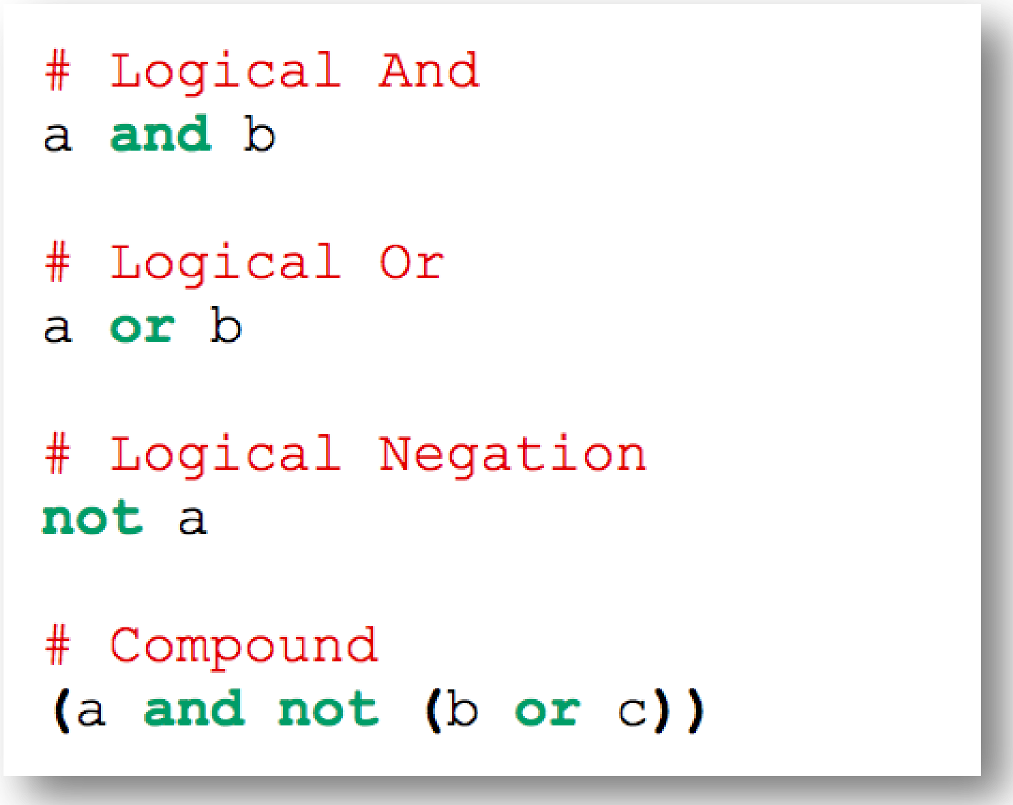
```
a = 10          # 10
a += 1          # 11
a -= 1          # 10

b = a + 1       # 11
c = a - 1       # 9

d = a * 2       # 20
e = a / 2       # 5
f = a % 3       # 1
g = a ** 2      # 100
```

String Manipulation



```
animals = "Cats " + "Dogs "
animals += "Rabbits"
# Cats Dogs Rabbits

fruit = ', '.join(['Apple', 'Banana', 'Orange'])
# Apple, Banana, Orange

date = '%s %d %d' % ('Sept', 11, 2010)
# Sept 11 2010

name = '%(first)s %(last)s' % {
    'first': 'Nowell',
    'last': 'Strite'}
# Nowell Strite
```

Logical Comparison

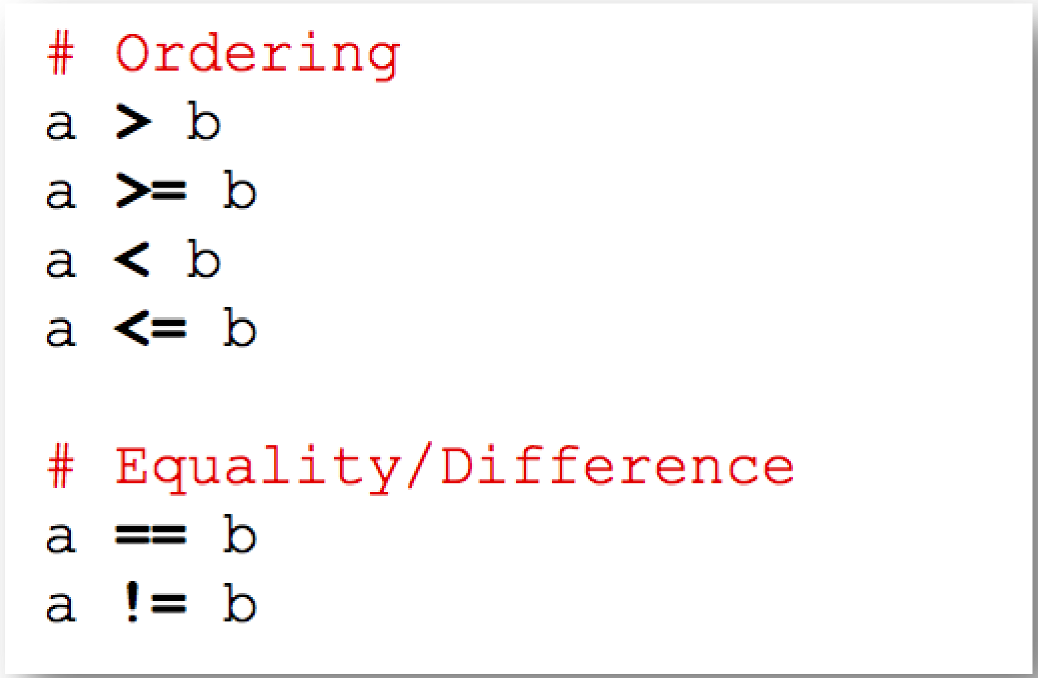


```
# Logical And
a and b

# Logical Or
a or b

# Logical Negation
not a

# Compound
(a and not (b or c))
```

Identity Comparison

```
# Identity
1 is 1 == True

# Non Identity
1 is not '1' == True

# Example
bool(1) == True
bool(True) == True

1 and True == True
1 is True == False
```

Arithmetic Comparison



```
# Ordering
a > b
a >= b
a < b
a <= b

# Equality/Difference
a == b
a != b
```

Expressions

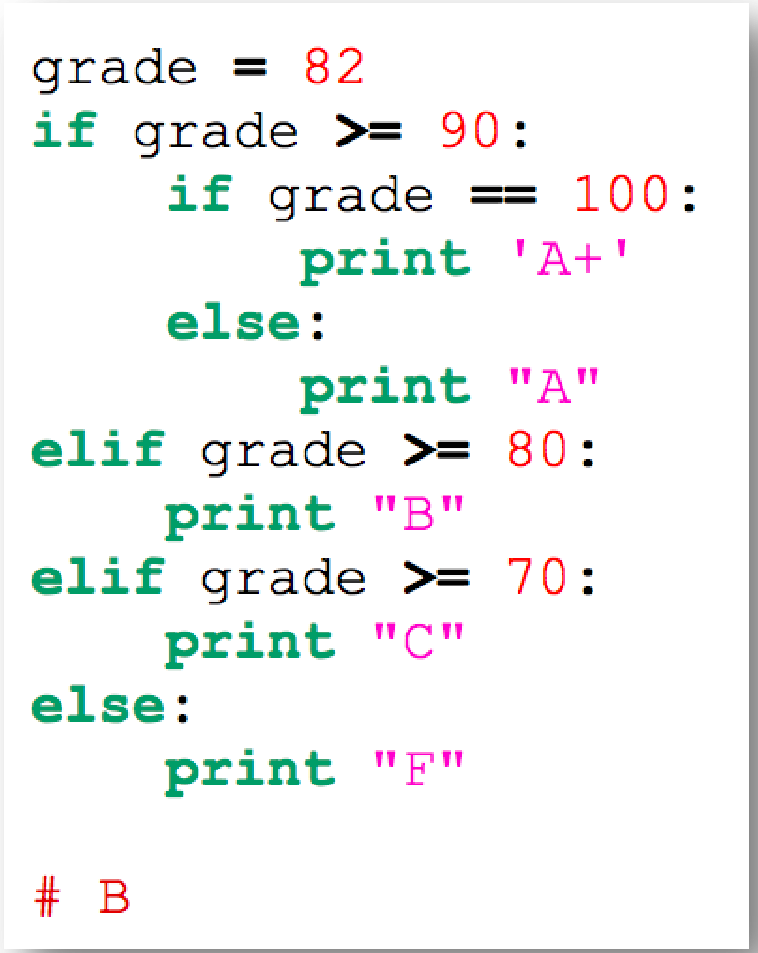
- Generally while writing expressions we put spaces before and after every operator so that the code becomes clearer to read.

Predict the Output

```
a = 9
b = 12
c = 3
x = a - b / 3 + c * 2 - 1
y = a - b / (3 + c) * (2 - 1)
z = a - (b / (3 + c) * 2) - 1
print("X = ", x)
print("Y = ", y)
print("Z = ", z)
```

Control Flow

Conditionals



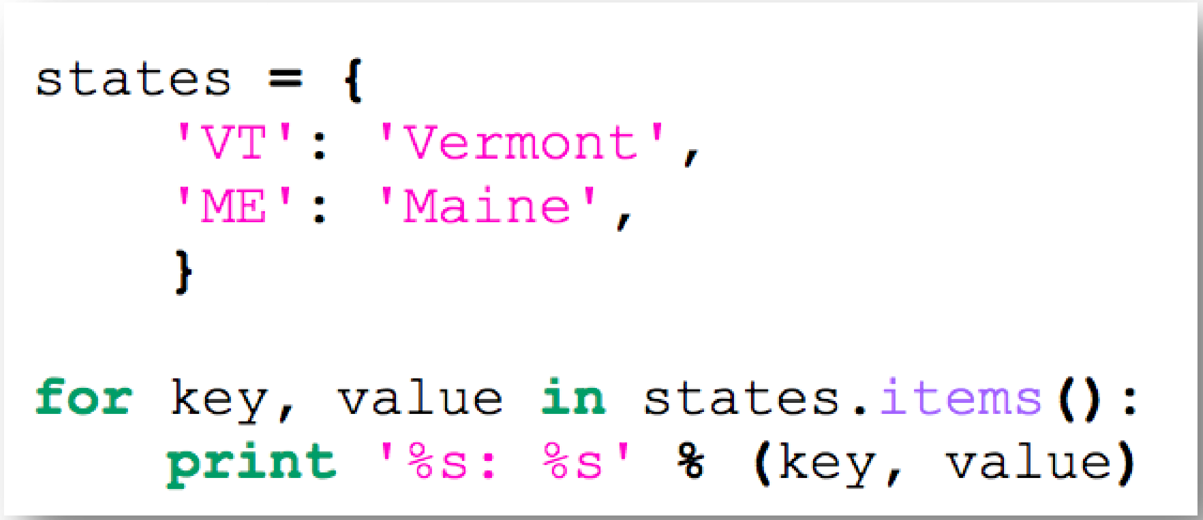
```
grade = 82
if grade >= 90:
    if grade == 100:
        print 'A+'
    else:
        print "A"
elif grade >= 80:
    print "B"
elif grade >= 70:
    print "C"
else:
    print "F"

# B
```

For Loop

```
for x in range(10): #0-9
    print x
```

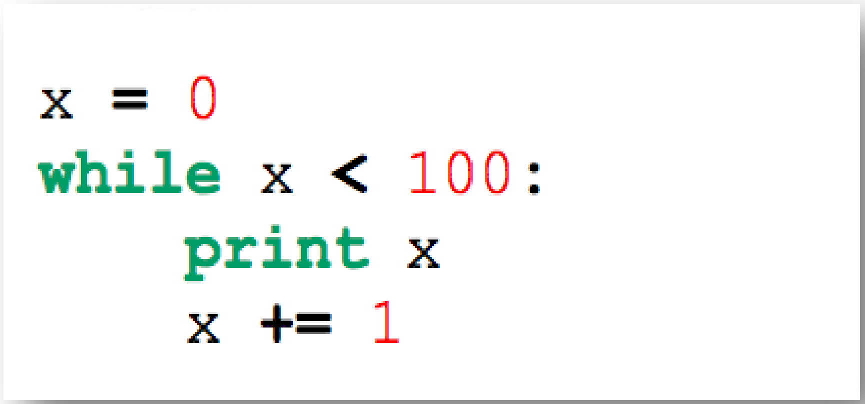
```
fruits = ['Apple', 'Orange']

for fruit in fruits:
    print fruit
```

Expanded For Loop



```
states = {
    'VT': 'Vermont',
    'ME': 'Maine',
    }

for key, value in states.items():
    print '%s: %s' % (key, value)
```

While Loop



```
x = 0
while x < 100:
      print x
      x += 1
```

List Comprehensions

- Useful for replacing simple for-loops.

```
odds = [ x for x in range(50) if x % 2 ]
```

```
odds = []
for x in range(50):
    if x % 2:
        odds.append(x)
```

Reading input from the Keyboard

```
number = int(input("Enter an integer: "))
if number < 100:
    print("Your number is smaller than 100")
else:
    print("Your number is greater than 100")
```

More Examples

```
amount = float(input("Enter amount: "))
inrate = float(input("Enter Interest rate: "))
period = int(input("Enter period: "))
value = 0
year = 1
while year <= period:
    value = amount + (inrate * amount)/100
    print("Year %d Rs. %.2f" % (year, value))
    amount = value
    year = year + 1
```

Let’s Practice

- Take inputs from user to calculate average of N numbers
- convert the given temperature to Celsius from Fahrenheit by using the formula C=(F-32)/1.8
- Provide marks of three subject and declare the result, result declaration is based on below conditions:
  - **Condition 1:** -All subjects marks is greater than 60 is Passed
  - **Condition 2:** -Any two subjects marks is greater than 60 is Promoted
  - **Condition 3:** -Any one subject marks is greater than 60 or all subjects’ marks less than 60 is failed.
- Find out all the ***Armstrong numbers*** *falling* in the range of **100-999**
- https://goo.gl/y30gYG

Functions in Python

Basic Function

```
#Syntax
def functionname(params):
    statement1
    statement2
```

```
def sum(a, b):
    return a + b
res = sum(234234, 34453546464)
print (res)
```

Lets Practies

- Write function to Find out ***Armstrong numbers .***
- Use **armstrong() to** Find out all the ***Armstrong numbers*** *falling* in the range of **100-999**

Local and global variables

```
def change(b):
    a = 90
    print(a)
a = 9
print("Before the function call ", a)

print("inside change function")
change(a)

print("After the function call ", a)
```

Local and global variables continue…

```
def change(b):
    global a
    a = 90
    print(a)
a = 9
print("Before the function call ", a)
print("inside change function")
change(a)
print("After the function call ", a)
```

Default argument value

- In a function variables may have default argument values, that means if we don’t give any value for that particular variable it will assigned automatically.

```
def test(a , b=-99):
    if a > b:
        return True
    else:
        return False
print test(12, 23)
print test(12)
```

Keyword arguments

```
def func(a, b=5, c=10):
    print('a is', a, 'and b is', b, 'and c is', c)

func(12, 24)
# a is 12 and b is 24 and c is 10

func(12, c = 24)
# a is 12 and b is 5 and c is 24

func(b=12, c = 24, a = -1)
# a is -1 and b is 12 and c is 24
```

Built-in Functions

Built-in Functions				
abs()	divmod()	input()	open()	staticmethod()
all()	enumerate()	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	filter()	len()	range()	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	reduce()	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	map()	repr()	xrange()
cmp()	globals()	max()	reversed()	zip()
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	
delattr()	help()	next()	setattr()	
dict()	hex()	object()	slice()	
dir()	id()	oct()	sorted()	

File handling

File I/O

- To open a file we use **open()** function. It requires two arguments, first the file path or file name, second which mode it should open.
- Modes are like
  - "r" -> open read only, you can read the file but can not edit / delete anything inside
  - "w" -> open with write power, means if the file exists then delete all content and open it to write
  - "a" -> open in append mode
- For Closing a file We use method **close()**

Read Example

```
fobj = open("sample.txt")
print fobj.read()
''' If you call read() again it will return empty string as it
already read the whole file. readline() can help you to read
one line each time from the file.'''
print fobj.readline()
fobj.close()
```

Write Demo

```
fobj = open("ircnicks.txt", 'w')
fobj.write('powerpork\n')
fobj.write('indrag\n')
fobj.write('mishti\n')
fobj.write('sankarshan')
fobj.close()
```

Using the with statement

```
with open('083FileIORealTimeDemo.py') as fobj:
    for line in fobj:
        print line,
```

Python File Modes

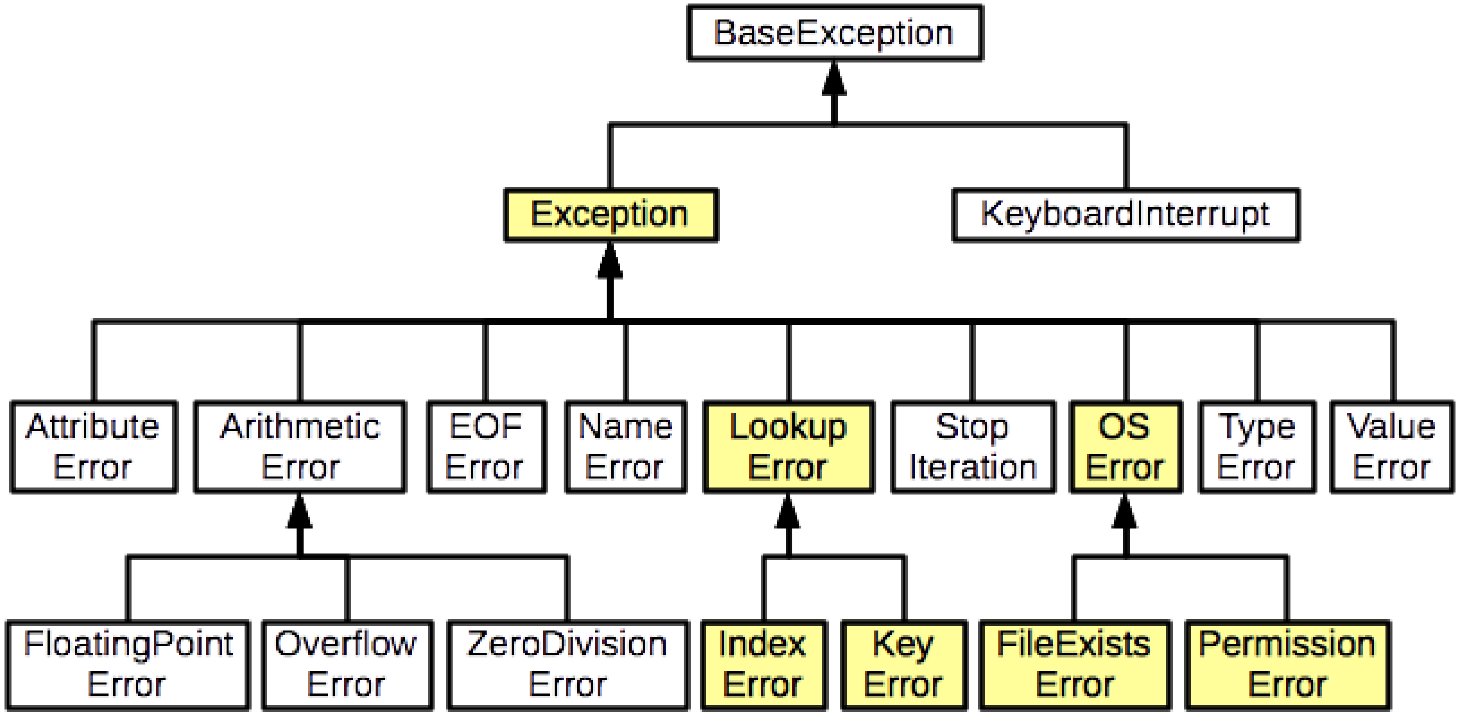
Mode	Description
'r'	Open a file for reading. (default)
'w'	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
'x'	Open a file for exclusive creation. If the file already exists, the operation fails.
'a'	Open for appending at the end of the file without truncating it. Creates a new file if it does not exist.
't'	Open in text mode. (default)
'b'	Open in binary mode.
'+'	Open a file for updating (reading and writing)

Exceptions

Types of Error in Python

- **NameError**
  - When one starts writing code, this will be one of the most command exception he/she will find. This happens when someone tries to access a variable which is not defined.
  - **Example :** **print** (Onkar)
- **TypeError**
  - TypeError is also one of the most found exception. This happens when someone tries to do an operation with different kinds of incompatible data types. A common example is to do addition of Integers and a string.
  - **Example :** **print** (1 + “Onkar")

Exception Hierarchy



How to handle exceptions?

Basic Syntax

```
try:
     statements to be inside try clause
     statement2
     statement3
except ExceptionName:
     statements to evaluated in case of ExceptionName happens
```

How Exception Handling Works in Python

- It works in the following way:
  - First all lines between try and except statements.
  - If ExceptionName happens during execution of the statements then except clause statements execute
  - If no exception happens then the statements inside except clause does not execute.
  - If the Exception is not handled in the except block then it goes out of try block.

Example

```
def get_number():
    "Returns a float number"
    number = float(input("Enter a float number: "))
    return number

while True:
    try:
        print(get_number())
    except TypeError:
        print("You entered a wrong value.")
    except:
         print("Unknown Error.")
```

Raising exceptions

```
try:
    raise ValueError("A value error happened.")
except ValueError:
    print("ValueError in our code.")
```

Using finally for cleanup

```python
try:
    fobj = open("hello.txt", "w")
    res = 12 / 0
except ZeroDivisionError:
    print("We have an error in division")
finally:
    fobj.close()
    print("Closing the file object.")
```

Classes & Objects

Your first class

Sytnax

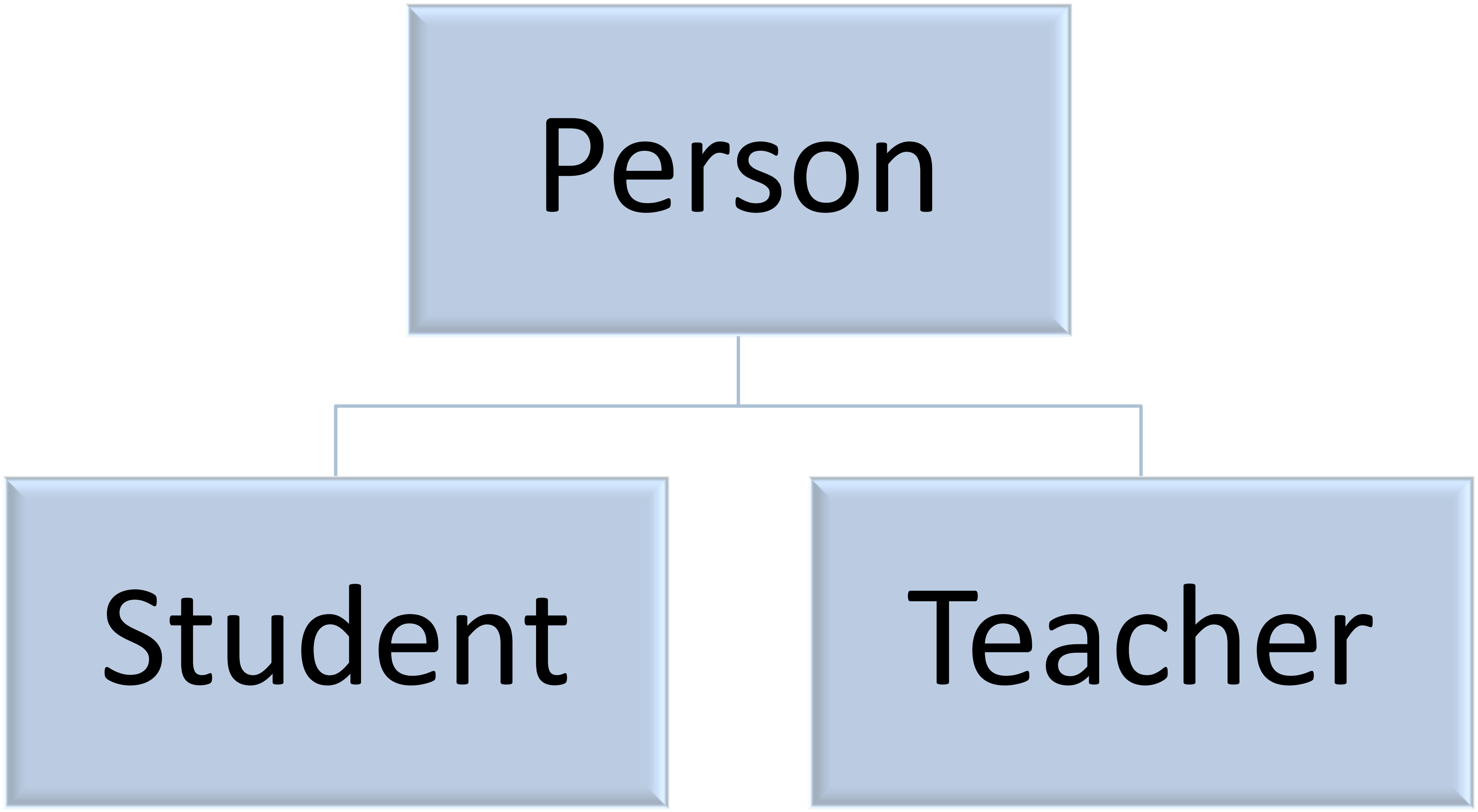
```
class nameoftheclass(parent_class):
    statement1
    statement2
    statement3


class MyClass(object):
    a = 90
    b = 88

p = MyClass()
print p
```

__init__ method

- __init__ is a special method in Python classes, it is the constructor method for a class.

```
class Student(object):
    ""“ Returns a ```Student``` object with the given name, branch and year.
    """
    def __init__(self, name, branch, year):
        self.name = name
        self.branch = branch
        self.year = year
        print("A student object is created.")

    def print_details(self):
        """
        Prints the details of the student.
        """
        print("Name:", self.name)
        print("Branch:", self.branch)
        print("Year:", self.year)

# std1 = Student()
std1 = Student('Onkar','CSE','2005')
std1.print_details()
```

Inheritance



Person class

```
class Person(object):
    """
    Returns a ```Person``` object with given name.
    """
    def __init__(self, name):
        self.name = name

    def get_details(self):
        "Returns a string containing name of the person"
        return self.name
```

Student class

```
class Student(Person):
    """
    Returns a ```Student``` object, takes 3 arguments, name,
branch, year.
    """
    def __init__(self, name, branch, year):
        Person.__init__(self, name)
        self.branch = branch
        self.year = year

    def get_details(self):
        "Returns a string containing student's details."
        return "%s studies %s and is in %s year." %
       (self.name, self.branch, self.year)
```

Teacher Class

```
class Teacher(Person):
    """
    Returns a ```Teacher``` object, takes a list of strings
(list of papers) as argument.
    """
    def __init__(self, name, papers):
        Person.__init__(self, name)
        self.papers = papers

    def get_details(self):
        return "%s teaches %s" % (self.name,
       ','.join(self.papers))
```

Execute code

```
person1 = Person('Sachin')
student1 = Student('Kushal', 'CSE', 2005)
teacher1 = Teacher('Prashad', ['C', 'C++'])
print(person1.get_details())
print(student1.get_details())
print(teacher1.get_details())
```

Multiple Inheritance

```
class MyClass(Parentclass1, Parentclass2,...):
       def __init__(self):
              Parentclass1.__init__(self)
              Parentclass2.__init__(self)
...
...
```

Getters and setters in Python

- One simple answer, **don’t.**
- Just use the **attributes** directly.

```
class Student(object):
    def __init__(self, name):
        self.name = name

std = Student("Kushal Das")
print(std.name)
std.name = "Python"
print(std.name)
```

Python's Way

- No interfaces
- No real private attributes/functions
- Private attributes start (but do not end) with double underscores.
- Special class methods start and end with double underscores.
  - __init__, __doc__, __cmp__, __str__

Modules

Introduction to Modules

- All the code we wrote in the Python interpreter was **lost when we exited the interpreter**. But when people write large programs they tend to break their code into **multiple different files** for ease of use, debugging and readability. In **Python we use modules** to achieve such goals

“bars.py” Module Example

```
"""
Bars Module
============
This is an example module with provide different ways to
print bars.
"""
def starbar(num):
    """Prints a bar with * :arg num: Length of the bar """
    print('*' * num)

def hashbar(num):
    """Prints a bar with # :arg num: Length of the bar """
    print('#' * num)

def simplebar(num):
    """Prints a bar with - :arg num: Length of the bar """
    print('-' * num)
```

Use bars.py Module

```python
import bars

bars.hashbar(10)

bars.simplebar(10)

bars.starbar(10)
```

Importing modules

- You can even import selected functions from modules.

```
from bars import simplebar, starbar
simplebar(20)
```

Default modules

- Now your Python installation comes with different modules installed, you can use them as required and install new modules for any other special purposes.

```
>>> help()

Welcome to Python 3.5's help utility!
If this is your first time using Python, you should definitely check out
the tutorial on the Internet at http://docs.python.org/3.5/tutorial/.
Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules. To quit this help utility and
return to the interpreter, just type "quit".
To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics". Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help> modules
```

Module os

- >>> import os
- >>> os.getuid()
- >>> os.getpid()
- >>> os.getppid()
- >>> os.uname()

Python Database communication

MySQLdb module

- The _mysql module implements the MySQL C API directly. It is not compatible with the Python DB API interface.

```
import MySQLdb as mdb
import sys

try:
    con = mdb.connect('localhost', 'root', 'root', 'testdb');

    cur = con.cursor()
    cur.execute("SELECT VERSION()")

    ver = cur.fetchone()

    print "Database version : %s " % ver
except mdb.Error, e:

    print "Error %d: %s" % (e.args[0],e.args[1])
    sys.exit(1)

finally:
     if con:
        con.close()
```

Creating and populating a table

```
import MySQLdb as mdb

con = mdb.connect('localhost', 'root', 'root', 'testdb');

with con:

    cur = con.cursor()
    cur.execute("DROP TABLE IF EXISTS Writers")
    cur.execute("CREATE TABLE Writers(Id INT PRIMARY KEY AUTO_INCREMENT, \
                 Name VARCHAR(25))")
    cur.execute("INSERT INTO Writers(Name) VALUES('Jack London')")
    cur.execute("INSERT INTO Writers(Name) VALUES('Honore de Balzac')")
    cur.execute("INSERT INTO Writers(Name) VALUES('Lion Feuchtwanger')")
    cur.execute("INSERT INTO Writers(Name) VALUES('Emile Zola')")
    cur.execute("INSERT INTO Writers(Name) VALUES('Truman Capote')")
```

Retrieving data

```
import MySQLdb as mdb

con = mdb.connect('localhost', 'root', 'root', 'testdb');

with con:

    cur = con.cursor()
    cur.execute("SELECT * FROM Writers")

    rows = cur.fetchall()

    for row in rows:
        print row
```

Retrieving data Continue…

```
import MySQLdb as mdb

con = mdb.connect('localhost', 'root', 'root', 'testdb');

with con:

    cur = con.cursor()
    cur.execute("SELECT * FROM Writers")

    rows = cur.fetchall()

    for row in rows:
        print row
```

The dictionary cursor

- There are multiple cursor types in the MySQLdb module. The default cursor returns the data in a tuple of tuples. When we use a dictionary cursor, the data is sent in a form of Python dictionaries.

Example

```
import MySQLdb as mdb

con = mdb.connect('localhost', 'root', 'root', 'testdb')

with con:

    cur = con.cursor(mdb.cursors.DictCursor)
    cur.execute("SELECT * FROM Writers LIMIT 4")

    rows = cur.fetchall()

    for row in rows:
        print row["Id"], row["Name"]
```

PreparedStatement

```
import MySQLdb as mdb

con = mdb.connect('localhost', 'root', 'root', 'testdb')

with con:

    cur = con.cursor()

    cur.execute("UPDATE Writers SET Name = %s WHERE Id = %s",
        ("Guy de Maupasant", "4"))

    print "Number of rows updated:",  cur.rowcount
```

That’s it for Today ☺

http://tutorialspoint.com/python/

https://wiki.python.org/moin/BeginnersGuide

http://learnpythonthehardway.org/book/index.html

http://www.tutorialspoint.com/python/python_database_access.htm

https://docs.mongodb.org/getting-started/python/client/