**1. When we should which type class declaration??**

class classname:

python 3 and 2.7 is valid.

for Inheritance python 2.7 is invalid(Multiple and Multi level Inheritance)

class classname():

python 3 and 2.7 is valid

for Inheritance python 2.7 is invalid (Multiple and Multi level Inheritance)

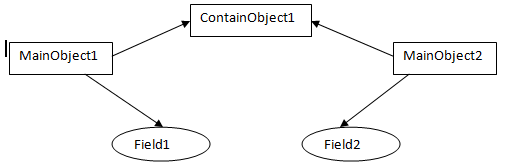
class classname(object):

python 3 and 2.7 is valid

**2. differences between deep and shallow copies with examples ??**

***What is Shallow Copy?***

Shallow copy is a bit-wise copy of an object. A new object is created that has an exact copy of the values in the original object. If any of the fields of the object are references to other objects, just the reference addresses are copied i.e., only the memory address is copied.

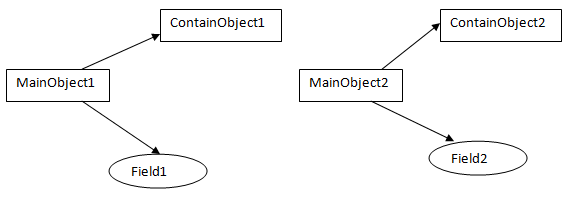


In this figure, the MainObject1 have fields "field1" of int type, and "ContainObject1" of ContainObject type. When you do a shallow copy of MainObject1, MainObject2 is created with "field3" containing the copied value of "field1" and still pointing to ContainObject1 itself. Observe here and you will find that since field1 is of primitive type, the values of it are copied to field3 but ContainedObject1 is an object, so MainObject2 is still pointing to ContainObject1. So any changes made to ContainObject1 in MainObject1 will reflect in MainObject2.

Now if this is shallow copy, lets see what's deep copy?

*What is Deep Copy?*

A deep copy copies all fields, and makes copies of dynamically allocated memory pointed to by the fields. A deep copy occurs when an object is copied along with the objects to which it refers.



In this figure, the MainObject1 have fields "field1" of int type, and "ContainObject1" of ContainObject type. When you do a deep copy of MainObject1, MainObject2 is created with "field3" containing the copied value of "field1" and "ContainObject2" containing the copied value of ContainObject1.So any changes made to ContainObject1 in MainObject1 will not reflect in MainObject2.

### Difference between Shallow and Deep Copy in Java

I think now we know what is deep and shallow copy of object in Java, let see some difference between them so that we can get some more clarity on them.

* When we call Object.clone(), this method performs a shallow copy of object, by copying data field by field, and if we override this method and by convention first call super.clone(), and then modify some fields to "deep" copy, then we get deep copy of object. This modification is done to ensure that original and cloned object are independent to each other.
* In shallow copy main or parent object is copied, but they share same fields or children if fields are modified in one parent object other parent fields have automatic same changes occur,but in deep copy this is not the case.
* If our parent object contains only primitive value then shallow copy is good for making clone of any object because in new object value is copied but if parent object contains any other object then only reference value is copied in new parent object and both will point to same object so in that case according to our need we can go for deep copy.
* Deep copy is expensive as compare to shallow copy in terms of object creation, because it involves recursive copying of data from other mutable objects, which is part of original object.

This is all about deep copy and shallow copy of objects in Java. Now the question comes when we use shallow copy and when go for deep copy , so answer would be  simple that if the object has only primitive fields or Immutable objects, then obviously we will go for shallow copy, but if the object has references to other mutable objects, then based on the requirement, shallow copy or deep copy can be chosen. Means  if the references are not modified anytime, then there is no point in going for deep copy, We can go for shallow copy. But if the references are modified often, then you need to go for deep copy. Again there is no hard and fast rule, it all depends on the requirement.

**3. Explain Checksum and Cyclic Redundancy Check ??**

Checksum and CRC

**Another secure-computing need is to ensure that the data has not been corrupted during transmission or encryption. There are a couple of popular ways to do this:**

**Checksum** - Probably one of the oldest methods of ensuring that data is correct, checksums also provide a form of authentication because an invalid checksum suggests that the data has been compromised in some fashion. A checksum is determined in one of two ways. Let's say the checksum of a packet is 1 [**byte**](http://computer.howstuffworks.com/bytes.htm) long. A byte is made up of 8 bits, and each bit can be in one of two states, leading to a total of 256 (28 ) possible combinations. Since the first combination equals zero, a byte can have a maximum value of 255.

* If the sum of the other bytes in the packet is 255 or less, then the checksum contains that exact value.
* If the sum of the other bytes is more than 255, then the checksum is the remainder of the total value after it has been divided by 256.

Let's look at a checksum example:

* **Bytes total 1,151**
* **1,151 / 256 = 4.496 (round to 4)**
* **4 x 256 = 1,024**
* **1,151 - 1,024 = 127 checksum**

**Cyclic Redundancy Check** (CRC) -

An error detection mechanism in which a special number is appended to a block of data in order to detect any changes introduced during storage (or transmission).

CRCs are similar in concept to checksums, but they use polynomial division to determine the value of the CRC, which is usually 16 or 32 bits in length. The good thing about CRC is that it is very accurate. If a single bit is incorrect, the CRC value will not match up. Both checksum and CRC are good for preventing random errors in transmission but provide little protection from an intentional attack on your data. Symmetric- and public-key encryption techniques are much more secure.

**4. Differences between TCP and UDP ??**

|  | TCP | UDP |
| --- | --- | --- |
| Acronym for | Transmission Control Protocol | User Datagram Protocol or Universal Datagram Protocol |
| Connection | TCP is a connection-oriented protocol. | UDP is a connectionless protocol. |
| Function | As a message makes its way across the [internet](http://www.diffen.com/difference/Internet_vs_World_Wide_Web) from one computer to another. This is connection based. | UDP is also a protocol used in message transport or transfer. This is not connection based which means that one program can send a load of packets to another and that would be the end of the relationship. |
| Usage | TCP is suited for applications that require high reliability, and transmission time is relatively less critical. | UDP is suitable for applications that need fast, efficient transmission, such as games. UDP's stateless nature is also useful for servers that answer small queries from huge numbers of clients. |
| Use by other protocols | HTTP, HTTPs, FTP, SMTP, Telnet | DNS, DHCP, TFTP, SNMP, RIP, VOIP. |
| Ordering of data packets | TCP rearranges [data](http://www.diffen.com/difference/Data_vs_Information) packets in the order specified. | UDP has no inherent order as all packets are independent of each other. If ordering is required, it has to be managed by the application layer. |
| Speed of transfer | The speed for TCP is slower than UDP. | UDP is faster because error recovery is not attempted. It is a "best effort" protocol. |
| Reliability | There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent. | There is no guarantee that the messages or packets sent would reach at all. |
| Header Size | TCP header size is 20 bytes | UDP Header size is 8 bytes. |
| Common Header Fields | Source port, Destination port, Check Sum | Source port, Destination port, Check Sum |
| Streaming of data | Data is read as a byte stream, no distinguishing indications are transmitted to signal message (segment) boundaries. | Packets are sent individually and are checked for integrity only if they arrive. Packets have definite boundaries which are honored upon receipt, meaning a read operation at the receiver socket will yield an entire message as it was originally sent. |
| Weight | TCP is heavy-weight. TCP requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control. | UDP is lightweight. There is no ordering of messages, no tracking connections, etc. It is a small transport layer designed on top of IP. |
| Data Flow Control | TCP does Flow Control. TCP requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control. | UDP does not have an option for flow control |
| Error Checking | TCP does error checking and error recovery. Erroneous packets are retransmitted from the source to the destination. | UDP does error checking but simply discards erroneous packets. Error recovery is not attempted. |
| Fields | 1. Sequence Number, 2. AcK number, 3. Data offset, 4. Reserved, 5. Control bit, 6. Window, 7. Urgent Pointer 8. Options, 9. Padding, 10. Check Sum, 11. Source port, 12. Destination port | 1. Length, 2. Source port, 3. Destination port, 4. Check Sum |
| Acknowledgement | Acknowledgement segments | No Acknowledgment |
| Handshake | SYN, SYN-ACK, ACK | No handshake (connectionless protocol) |

**5. How do you know which ports are opened in your machine.??**

## **netstat command to find open ports**

The syntax is:  
# netstat --listen  
OR  
# netstat -l

To display open ports and established TCP connections, enter:  
$ netstat -vatn  
To display only open UDP ports try the following command:  
$ netstat -vaun  
If you want to see FQDN (full dns hostname), try removing the -n flag:  
$ netstat -vat  
FreeBSD/OS X Unix user try the following command:  
$ netstat -na | grep -i LISTEN  
$ netstat -f inet -na | grep -i LISTEN

## lsof Command Examples

To display the list of open ports, enter:  
# lsof -i  
To display all open files, use:  
# lsof  
To display all open IPv4 network files in use by the process whose PID is 9255, use:  
# lsof -i 4 -a -p 9255  
Another example:  
# lsof -iTCP -sTCP:LISTEN

**6.What are the built-in type does python provides?**

There are mutable and Immutable types of Pythons built in types Mutable built-in types

* List
* Sets
* Dictionaries

Immutable built-in types

* Strings
* Tuples
* Numbers

**7. What are Decorators in python ,explain with example ?**

Decorators provide a simple syntax for calling higher-order functions. By definition, a decorator is a function that takes another function and extends the behavior of the latter function without explicitly modifying it.

def p\_decorate(func):

def func\_wrapper(name):

return "<p>{0}</p>".format(func(name))

return func\_wrapper

@p\_decorate

def get\_text(name):

return "lorem ipsum, {0} dolor sit amet".format(name)

print get\_text("John")

# Outputs <p>lorem ipsum, John dolor sit amet</p>