# CS 407
# Programming 1 + Grad Programming 1
# Due: Sunday of Week 5 11:59pm (via Gradescope)

This assignment may be completed individually or in groups of 2. Your code for files you should edit, `NAMEstd.py`, `NAMEtyrant.py`, `NAMEpropshare.py`, and `NAMEtourney.py` and your writeup `NAME.pdf` (where NAME is replaced by your team name) should be submitted via Gradescope. Please include **only** these python files, not the rest of the code we provided you. Please ignore any complaints from Gradescope about an autograder; this assignment does not have one.
While you are permitted to discuss your clients in general terms with other students as much as you like, each group must write their own code and explanations. If you want a partner and don't have one, post to Piazza. Both partners must contribute to all aspects of the assignment.

## Goal

In this assignment you will program a number of clients for the BitTorrent protocol and test them in a simulation environment. You will be evaluated based on the code you submit and a writeup

## Setup

**Tools:** Make sure Python 3 works on your computer system (`http://python.org/download/`).

**Obtaining materials:** Download the BitTorrent .zip-archive from Blackboard and make sure you have all the relevant files: `dummy.py`, `lessdummy.py`, `history.py`, `messages.py`, `peer.py`, `seed.py`, `sim.py`, `start.py`, `stats.py`, `util.py`.

**Generating .py-files:** Pick a cool group name, perhaps based on your initials, so that it will be unique in the class. Run `python start.py NAME`, substituting your group name for `NAME`. This will create appropriately named template files for your clients. For example, if your group name was "kash":

```
> python start.py kash
Copying lessdummy.py to kashstd.py...
Copying lessdummy.py to kashtyrant.py...
Copying lessdummy.py to kashpropshare.py...
Copying lessdummy.py to kashtourney.py...
All done.  Code away!
```

In each of the files, you will need to change `LessDummy` in line 13 (where it says `class Dummy(Peer)`) to your group name and the client specification (e.g. `KashStd`, `KashTyrant`, `KashPropShare`, `KashTourney`).

**The simulator:** You are given a BitTorrent simulator. It has the following general structure:

- Time proceeds in integer rounds, starting at 0 (0, 1, 2, ...).
- There is a single file being shared, composed of `num-pieces` pieces, each of which is composed of `blocks-per-piece` blocks.
- There is a set of peers, which should include at least 1 seed. Seeds start with all the pieces of the file and have the maximum upload bandwidth. Other peers start with none of the pieces.
- Each peer has an upload bandwidth, measured in blocks per round. This is chosen uniformly at random from a specified range (except for seeds, who have the maximum bandwidth). Download bandwidth is unlimited.

- Each round proceeds as follows:
  1. Each peer must provide a list of `request` objects, each asking for blocks from a particular piece from a particular other peer. This is the `requests`-function.
  2. Each peer is passed the requests it has received in this round, and asked to return a list of `upload` obejcts which specify how much of its upload bandwidth to allocate to any given peer. This is the `uploads`-function.
  3. These lists are checked for consistency, i.e. requests must ask for pieces the peer has, uploads must add up to no more than the peer's bandwidth cap, etc.
  4. Peers who are being uploaded to get their blocks. Multiple requests are satisfied in order until the shared bandwidth is exhausted. For example, if peer 1 requests the last 7 blocks of piece 12, and also piece 11 starting at block 0, and peer 2 includes `Upload(from_id=2, to_id=1, bw=10)` in its list of uploads, peer 1 will get the last 7 blocks of piece 12, and the first 3 of piece 11.
  5. The events of this round are saved in a `history` object, which is available in future rounds.
- The simulation ends once all peers have all the pieces or the maximum number of rounds `max-round` is exceeded.

**Building clients:** Familiarize yourself with the provided code. You should not need to change any of the provided files, only modify the ones created by `start.py`. The only files you should need to read in any detail are `messages.py`, `dummy.py`, and `lessdummy.py`, along with a skim of `history.py`.

**Running tests:** Here is an initial test command to run. It sets a bunch of simulation parameters, and creates 1 seed and 2 dummy clients. Note that as these commands have many arguments you may need to paste each line of the command separately to avoid including the linebreak and having the second line of arguments ignored.

```
> python sim.py --loglevel=debug --num-pieces=2 --blocks-per-piece=2
  --min-bw=1 --max-bw=3 --max-round=5 Seed,1 Dummy,2
```

Make sure you understand the output.
`> python sim.py -h` will show all the command line options.
When analyzing the performance of your clients, you can set the log level to `info`. This logs much less information so will be much faster. Use `--iters=...` to extract statistics for multiple runs of the same setup.

**Improving Dummy:** The dummy client is not very smart; it provides just enough to successfully download the file. Run the following command. From the summary stats at the end, you'll see leechers finishing, on average, in around 21 or 22 rounds.

```
python sim.py --loglevel=info --num-pieces=8 --blocks-per-piece=32
--min-bw=16 --max-bw=64 --max-round=1000 --iters=12 Seed,1 Dummy,20
```

Now, run the following command, which replaces the Dummy client with the LessDummy client which implements rarest first piece selection.

```
python sim.py --loglevel=info --num-pieces=8 --blocks-per-piece=32
--min-bw=16 --max-bw=64 --max-round=1000 --iters=12 Seed,1 LessDummy,20
```

You should see the completion time drop, perhaps to around 18 or 19 rounds on average. Compare the code in `dummy.py` and `lessdummy.py` to see the code changes that enabled this. These are the sorts of code changes you will have to make in the course of the assignment.

**Analyzing Clients:** For the parts of the assignment which ask you to analyze a situation and report statistics, a good command for your analysis may be

```
python sim.py --loglevel=info --num-pieces=128 --blocks-per-piece=32
  --min-bw=16 --max-bw=64 --max-round=1000 --iters=32 Seed,2 NAMEstd,10
```

but we encourage you to try varying parameters. The following command will run faster and is more useful while debgging early on.

```
python sim.py --loglevel=info --num-pieces=128 --blocks-per-piece=32
  --min-bw=16 --max-bw=64 --max-round=1000 --iters=12 Seed,2 NAMEstd,3
```

# Assignment

Satisfactory completion of each of the following four tasks is worth 0.5 points. *The expectation is that undergraduates will do 1 and either 2 or 3 for their 1 point and graduate students will do all 4 for their 2 points, but undergraduates are welcome to do more for bonus points.* In addition, there are bonus points available in the tournament, as described in the fifth task.
Grading will primarily based on your writeup, with your code checked to make sure it is consistent with your writeup and provides a plausible implementation of the relevant type of client. In your writeup you will be expected to describe your assumptions / design decisions as well as perform some analysis. Provide your answers based on test runs with the clients you programmed. When asked for comparative performance results, provide some evidence (e.g., simple statistics) for the results you are reporting/describing.

1. Task 1: Reference Client (0.5 points)

   Implement the BitTorrent *reference client* as described in Chapter 5, including rarest-first, recipocation and optimistic unchoking. This should be class `TeamnameStd` in `teamnamestd.py`. Not all the details are in Chapter 5, so you will have to make some assumptions. Explain all of the assumptions you make in your writeup.

   To provide some additional detail about the code in this file, there are three methods (don't forget to change the class name from LessDummy to the appropriate class name for your team!). The first, `post_init`, requires very little code to be added, particularly for the reference client, but is where you should declare and initialize any variables or datastructures you need to track the history to implement your client's strategy. The second, `requests`, is where your client determines which pieces to request from others. Dummy requested pieces at random. We implemented rarest first in LessDummy and the setup code copied our implemention for you. You don't hae to change anything here, but there are certainly cleverer versions you can consider than what we have done. The third, `uploads`, is where most of the work is and represents the unchoking policy. There is currently an extremely simple version where a single peer is chosen at random to be unchoked in each round. You need to modify this to implement the reference client's unchoking behavior as discussed in class / the text. To get a sense of how you might do this, you could compare Dummy and LessDummy to see how we modified things to implement rarest first. You'll need a similar or perhaps slightly larger scale of modifications in implementing proper unchoking behavior.

   When you move on to subsequent tasks, you'll likely be able to reuse at least some of your implementation of `uploads`.

2. Task 2: BitTyrant Client (0.5 points)

   Implement the *BitTyrant client*. This should be class `TeamnameTyrant` in `teamnametyrant.py`. You may have to introduce additional accounting and book-keeping procedures beyond what is provided by `peers` and `history`. In your writeup, provide the results of running a simulation with a single BitTyrant client among reference clients. How does the performance of BitTyrant compare to the reference client?

3. Task 3: Proportional Sharing Client (0.5 points)

   Implement the *PropShare client*. This should be class `TeamnamePropShare` in `teamnamepropshare.py`. The *PropShare client* allocates upload bandwidth based on the downloads received from peers in the

previous round: It calculates what share each peer contributed to the total download and allocates its own bandwidth proportionally. In addition it reserves a small share of its bandwidth for optimistic unchoking (e.g., 10%). For example

- In round $k$ the client received $4, 6, 1, 9$ blocks from peers $A, B, C, D$, respectively
- In round $k+1$ peers $A, B, E, F$ request pieces from the client
- The client allocates $\frac{4}{4+6} \cdot 90\% = 36\%$ and $\frac{6}{4+6} \cdot 90\% = 54\%$ of it's upload bandwidth to $A$ and $B$, respectively
- $E$ is randomly selected and allocated 10%

In your writeup, provide the results of running a simulation with a single PropShare client among standard clients. How does the performance of PropShare compare to the standard client?

4. Task 4: Additional Analysis (0.5 points)

   Design and run some experiments to answer the following questions. Explain your setup, and the results for each.

   (a) How does a population of only *BitTyrant* clients perform?

   (b) How does a population of only *PropShare* clients perform?

   (c) What is the overall performance and relative performance of each client in a population with all three types?

   (d) Write a paragraph about what you learned from these exercises about BitTorrent, game theory, and programming strategic clients? (We aren't looking for any particular answers here, but are looking for evidence of real reflection.)

5. Task 5: Class Tournament (Bonus!)

   Write a client for the class competition in class `TeamnameTourney` in `teamnametourney.py`. We will run the tournament in a neighborhood containing one instance of each group's tournament client and 2 seeds. This setup will be run a number of times times. Winners will be determined by the average time to complete the download of the entire file. Likely parameters for the competition are

   ```
   python sim.py --loglevel=info  --num-pieces=128 --blocks-per-piece=32
     --min-bw=16 --max-bw=64 --max-round=1000 --iters=256 Seed,2 Client1,1
     Client2,1 ...
   ```

   but they may vary depending on runtime of the clients.
   Your client can access the simulation configuration via `self.conf` (see the bottom of `sim.py` for the available parameters), so you shouldn't hard-code any of these values.

   The top 5 overall entries will receive 0.5 bonus points. The top graduate student entry and the top undergraduate entry will receive an additional 0.5 bonus points. If an undergraduate and graduate student work together they will be in the graduate category for this purpose.

# Comments and Hints

**Python features:** Useful python features / functions / modules that you may want to google and maybe use:

- random: shuffle, choice, sample
- min, max, map, filter, zip
- set
- list and dictionary comprehensions

**Programming in Python:** For general information on Python and tutorials, check out `http://docs.python.org/tutorial/`.

**Design:**
- Make sure you sort things correctly (you'll often want decreasing instead of the default increasing)
- Beware of symmetry. All peers know about all the others, and there's no lying about available pieces modeled, so silly things can happen; e.g., if all your tit-for-tat peers use the same deterministic algorithm for choosing which piece to request next, you may have a situation where everyone always has the same pieces, there's nothing to trade, and the system devolves to very slow downloading of everything from the seed.
- Debug using small numbers of peers, pieces, and blocks so you can see what's happening. Realize that the relative performance of different strategies is different at very small scales (e.g. tit-for-tat will not kick in if you only have 2 pieces).
- Look at more than one round of history when making peering decisions. Otherwise thrashing will occur (I upload to you in round 1. You didn't upload to me. So in round 2 you upload to me to reciprocate, but I've already moved on to someone else).

**Debugging:** Make sure your clients work with the unmodified simulator. If you changed anything, redownload the simulator and double check that your clients work with it.

**Manipulations:** The code is designed so that it's very hard to mess up the main simulation accidentally, but because everything is in the same process, it is still possible to cheat by directly modifying the simulation data structures and such. **Don't!**
Also don't use the specific ids of peers in any of your decision making, e.g.
`if peer.id.startswith("MyClient"): special-case-code`.

**Bugs:** If you find bugs in the code, let us know.

**Benchmark results** Please see some benchmarks below from a test of the staff implementations. Of course individual performance depends on your design decisions. These will test each three implementations individually. Run the last command if you have done all three (Std, Tyrant, and PropShare).

To test Std:

python sim.py –loglevel=info –num-pieces=128 –blocks-per-piece=32 –min-bw=16 –max-bw=64 –max-round=1000 –iters=5 Seed,2 Dummy,5 StaffStd,5

======== SUMMARY STATS ========
Uploaded blocks: avg (stddev)
Dummy2: 2179.8 (462.0)
Dummy3: 2454.2 (692.3)
StaffStd2: 2468.8 (869.1)
Dummy1: 2640.6 (668.3)
Dummy4: 2641.2 (668.5)
StaffStd1: 2644.2 (806.6)
Dummy0: 2788.6 (660.3)
StaffStd0: 3468.6 (945.2)
StaffStd4: 3487.6 (762.4)
StaffStd3: 3564.2 (162.3)
Seed1: 6258.6 (217.2)
Seed0: 6363.6 (254.8)
Completion rounds: avg (stddev)
Seed0: 0.0 (0.0)
Seed1: 0.0 (0.0)
StaffStd0: 95.6 (5.2)
StaffStd4: 96.0 (4.2895221179054435)
StaffStd1: 97.0 (2.1908902300206643)

StaffStd3: 97.2 (3.370459909270543)
StaffStd2: 99.6 (6.086049621881176)
Dummy3: 101.4 (4.586937976471886)
Dummy1: 101.6 (4.127953488110059)
Dummy0: 102.0 (3.521363372331802)
Dummy4: 102.0 (4.560701700396552)
Dummy2: 102.4 (3.7202150475476548)

To test Tyrant:

python sim.py –loglevel=info –num-pieces=128 –blocks-per-piece=32 –min-bw=16 –max-bw=64 –max-round=1000 –iters=5 Seed,2 Dummy,5 StaffTyrant,5

======== SUMMARY STATS ========
Uploaded blocks: avg (stddev)
StaffTyrant2: 2145.8 (321.3)
Dummy0: 2210.2 (450.7)
StaffTyrant4: 2306.4 (680.5)
StaffTyrant0: 2433.4 (899.4)
StaffTyrant1: 2730.0 (1268.3)
Dummy2: 2768.2 (761.6)
Dummy3: 2841.4 (1388.8)
Dummy1: 3280.2 (765.9)
StaffTyrant3: 3339.6 (735.0)
Dummy4: 3414.4 (939.8)
Seed1: 6648.2 (476.2)
Seed0: 6842.2 (529.8)
Completion rounds: avg (stddev)
Seed0: 0.0 (0.0)
Seed1: 0.0 (0.0)
StaffTyrant3: 96.4 (10.61319932913728)
StaffTyrant1: 103.0 (11.045361017187261)
StaffTyrant2: 103.2 (7.807688518377254)
StaffTyrant4: 104.0 (12.116104984688768)
Dummy1: 104.2 (10.283968105745952)
Dummy4: 105.8 (7.30479294709987)
StaffTyrant0: 107.0 (10.353743284435827)
Dummy2: 107.2 (8.863407922464136)
Dummy3: 107.6 (8.867919710958146)
Dummy0: 108.4 (7.657675887630659)

To test PropShare:

python sim.py –loglevel=info –num-pieces=128 –blocks-per-piece=32 –min-bw=16 –max-bw=64 –max-round=1000 –iters=5 Seed,2 Dummy,5 StaffPropShare,5

======== SUMMARY STATS ========
Uploaded blocks: avg (stddev)
StaffPropShare1: 2108.6 (990.4)
StaffPropShare4: 2388.0 (1044.5)
StaffPropShare3: 2435.0 (836.5)
Dummy4: 2668.0 (964.2)
Dummy3: 2709.2 (772.4)
StaffPropShare0: 2780.4 (869.0)
Dummy1: 2790.2 (888.5)
Dummy0: 3018.8 (546.5)

Dummy2: 3120.6 (355.5)
StaffPropShare2: 3648.4 (693.6)
Seed1: 6562.2 (238.6)
Seed0: 6730.6 (323.4)
Completion rounds: avg (stddev)
Seed0: 0.0 (0.0)
Seed1: 0.0 (0.0)
StaffPropShare2: 96.4 (3.7202150475476548)
StaffPropShare0: 97.0 (4.242640687119285)
StaffPropShare3: 97.2 (4.955804677345547)
StaffPropShare1: 100.8 (5.15363949069005)
StaffPropShare4: 101.0 (7.58946638440411)
Dummy3: 106.0 (3.3466401061363023)
Dummy2: 106.8 (3.1874754901018454)
Dummy4: 107.0 (5.0990195135927845)
Dummy1: 107.8 (4.749736834815167)
Dummy0: 108.6 (5.4626001134990645)

To test all three:

python sim.py –iters=20 –loglevel=info –num-pieces=32 –blocks-per-piece=32 –min-bw=32 –max-bw=64 –max-round=1000 Seed,2 Dummy,10 StaffTyrant,10 StaffStd,10 StaffPropShare,10

For Dummy, results were:

======== SUMMARY STATS ========
Uploaded blocks: avg (stddev)
Dummy9: 546.9 (119.7)
Dummy7: 549.8 (133.3)
Dummy5: 583.8 (98.1)
Dummy8: 619.0 (105.8)
Dummy6: 645.6 (128.1)
Dummy4: 710.2 (131.7)
Dummy3: 733.8 (103.7)
Dummy2: 765.1 (127.5)
Dummy1: 768.1 (123.7)
Dummy0: 817.6 (125.7)
Seed1: 1716.8 (76.6)
Seed0: 1783.2 (63.9)
Completion rounds: avg (stddev)
Seed0: 0.0 (0.0)
Seed1: 0.0 (0.0)
Dummy6: 29.15 (1.15217186218)
Dummy1: 29.2 (1.20830459736)
Dummy2: 29.2 (0.871779788708)
Dummy7: 29.25 (0.942072184071)
Dummy9: 29.35 (1.31434394281)
Dummy8: 29.4 (0.916515138991)
Dummy4: 29.45 (1.11691539518)
Dummy3: 29.5 (1.36014705087)
Dummy5: 29.55 (1.02347447452)
Dummy0: 29.65 (1.27573508222)

For Std, results were:

======== SUMMARY STATS ========

Uploaded blocks: avg (stddev)
StaffStd9: 603.9 (145.2)
StaffStd2: 616.1 (178.8)
StaffStd7: 618.9 (212.7)
StaffStd8: 622.9 (147.5)
StaffStd6: 631.5 (149.2)
StaffStd5: 665.7 (183.4)
StaffStd3: 702.8 (198.6)
StaffStd4: 717.4 (201.1)
StaffStd0: 785.4 (243.6)
StaffStd1: 787.5 (196.3)
Seed1: 1651.3 (85.4)
Seed0: 1836.9 (77.5)
Completion rounds: avg (stddev)
Seed0: 0.0 (0.0)
Seed1: 0.0 (0.0)
StaffStd1: 26.4 (2.00997512422)
StaffStd4: 27.1 (1.92093727123)
StaffStd5: 27.25 (2.14184499906)
StaffStd9: 27.6 (1.90787840283)
StaffStd0: 27.85 (2.17428149052)
StaffStd2: 27.85 (1.62095650774)
StaffStd8: 27.9 (2.25610283454)
StaffStd3: 28.0 (1.8973665961)
StaffStd7: 28.3 (1.70587221092)
StaffStd6: 28.4 (1.52970585408)

For tyrant:

======== SUMMARY STATS ========
Uploaded blocks: avg (stddev)
StaffTyrant7: 551.6 (205.7)
StaffTyrant6: 592.5 (170.9)
StaffTyrant3: 630.3 (165.6)
StaffTyrant4: 633.1 (195.7)
StaffTyrant5: 661.1 (171.0)
StaffTyrant8: 678.4 (158.8)
StaffTyrant9: 679.0 (177.1)
StaffTyrant0: 754.8 (193.4)
StaffTyrant1: 765.3 (184.1)
StaffTyrant2: 825.5 (196.5)
Seed1: 1628.8 (81.8)
Seed0: 1839.7 (77.3)
Completion rounds: avg (stddev)
Seed0: 0.0 (0.0)
Seed1: 0.0 (0.0)
StaffTyrant9: 26.35 (2.26439837484)
StaffTyrant2: 26.6 (2.0832666656)
StaffTyrant8: 26.75 (1.78535710714)
StaffTyrant3: 26.95 (1.98683164863)
StaffTyrant1: 27.0 (2.21359436212)
StaffTyrant0: 27.05 (2.39739441895)
StaffTyrant4: 27.4 (2.33238075794)
StaffTyrant5: 27.45 (2.7654113618)

StaffTyrant6: 27.8 (2.85657137142)
StaffTyrant7: 27.95 (2.51942453747)

And for PropShare:

======== SUMMARY STATS ========
Uploaded blocks: avg (stddev)
StaffPropShare2: 612.5 (196.3)
StaffPropShare6: 613.8 (146.2)
StaffPropShare3: 614.0 (182.5)
StaffPropShare4: 624.7 (182.8)
StaffPropShare9: 676.0 (143.2)
StaffPropShare5: 677.8 (185.0)
StaffPropShare8: 679.1 (215.8)
StaffPropShare1: 692.2 (192.0)
StaffPropShare7: 727.8 (174.3)
StaffPropShare0: 752.2 (190.5)
Seed1: 1690.4 (83.5)
Seed0: 1879.4 (97.2)
Completion rounds: avg (stddev)
Seed0: 0.0 (0.0)
Seed1: 0.0 (0.0)
StaffPropShare0: 27.3 (2.5903667694)
StaffPropShare7: 27.6 (2.57681974535)
StaffPropShare1: 28.15 (1.98179211826)
StaffPropShare5: 28.2 (2.27156333832)
StaffPropShare6: 28.35 (2.63201443765)
StaffPropShare9: 28.35 (2.30813777752)
StaffPropShare8: 28.6 (2.10713075057)
StaffPropShare3: 28.75 (2.75454170417)
StaffPropShare2: 28.8 (2.58069758011)
StaffPropShare4: 29.0 (1.92353840617)