* What is it?
  + Disco Bot is a Discord chat bot. It will specialize in LoL information, but will also have features relating to music and D&D.
* Languages, Libraries, APIs, Software
  + General
    - Python 3.6+
  + Discord
    - Discord.py
  + LoL
    - Lolwatcher.py
    - Py\_gg
* Features
  + Core
    - Play music and create playlists from youtube.
    - Search for player info, lol static data info, and lol statistics.
    - Roll die.
  + Extra
    - Ultimate Bravery for LoL
* Breakdowns
  + Cogs – Cogs are specific components of the bot that relate to an entire area.
    - LoL – Deals specifically with LoL. No other cog will deal with LoL.
    - Music – Music goes through this cog.
    - D&D – D&D utility stuff.
  + Commands
    - LoL
      * Player – Gets player info and ranks.
        + Usage: \player name –r=region
        + Needs:

summoner.by\_name

Gets summoner info by summoner name

league.positions\_by\_summoner

Gets rank info by summoner id

* + - * + Display:

(By Link) Profile Icon

Name, Level

For each Queue:

Rank Name, Division

LP, Ratio, Four Bonuses

* + - * Matchlist – Gets at most 20 recent matches for player.
        + Usage: \matchlist name –r=region –k=number
        + Needs:

summoner.by\_name

match.matchlist\_by\_account\_recent

Gets recent match list by account id

* + - * + Display:

Name, Region

For each Match:

Victory or Defeat

Match ID

Season, Queue

Champion, Lane

KDA

CS, CC, Vision, Other stuff (maybe Healing, Damage Reduced)

Note: For certain Queues, like ARAM, Lane does not matter so do not display it.

* + - * Match – Gets match info for match id
        + Usage: \match id –r=region –v

-v enables differential stats

* + - * + Needs:

Match.by\_id

Get match info by match id

* + - * + Match -> Teams -> Players -> Items, Runes, Deltas
      * Timeline – Gets match timeline for match id
        + Usage: \timeline id –r=region
        + Needs:

Match.timeline\_by\_match

Get match timeline by match id

NOTE: Not all matches have timelines

* + - * + Display:

For each Event:

Time Stamp

Any notable events:

Champion Kills:

Killer -> Victim

Any Assists

Building Kills:

Team/Killer -> Team Building

Any Assists

Epic Monster Kills:

Team/Killer -> Epic Monster

Any Assists

* + - * Mastery – Gets the top K mastery champions for a player.
        + Usage: \mastery name –r=region –k=number
        + Needs:

Championmastery.by\_summoner

Gets mastery entries sorted by number of points by summoner id

* + - * + Display:

Name

For each mastery entry:

Champion, Points, Level

* + - * Mastery – Gets a specific champion mastery for a player.
        + Usage: \mastery name champion –r=region
        + Needs:

Championmastery.by\_summoner\_by\_champion

Gets mastery entry by summoner id and champion id

* + - * + Display:

Name

Champion, Points, Level

* + - * Total Mastery – Gets total mastery score for a player.
        + Usage: \totalmastery name –r=region
        + Needs:

Championmastery.scores\_by\_summoner

Gets mastery total score by summoner id

* + - * + Display:

Name

Score

* + - * Challengers – Gets the top k challengers in a given queue
        + Usage: \challengers queue –r=region –k=number –v

-v means to show player info as well

* + - * + Needs:

League.challenger\_by\_queue

Gets all challengers by queue

Summoner.by\_name

If –v is enabled

* + - * + Display:

For each challenger:

Name

If –v:

Level, Icon

For each Queue:

Rank Division, Name

LP

Ratios, Four Bonuses

Else:

LP

Ratios, Four Bonuses

* + - * Masters – Gets the top k masters in a given queue
        + Usage: \masters queue –r=region –k=number –v
        + Note: Exact same as challengers, except uses League.master\_by\_queue instead.
      * Status - Gets the server status for a region
        + Usage: \status region
        + Needs:

Lolstatus.shard\_data

Gets server data by region

* + - * + Display:

For each slug (game, store, website):

Slug: Status

* + - * Spectate – Gets encryption key and how to for spectating a current live game
        + Usage: \spectate name –r=region
        + Needs:

Spectator.by\_summoner

Gets encryption key by summoner id

* + - * + Display:

HOW TO: + Encryption Key

Game Type, Map, Mode, Queue

For each Team:

For each Player:

Name, Champion

Rune Setup

Summoner Spells

Is a Bot?

* + - * Champion – Get champion info
        + Usage: \champion name –l –t –s –p=patch

-l: enables lore

-t: enables tips

-s: enables skins

* + - * + Needs:

Datadragon url for champion master list, champion, splash

* + - * + Display:

Link to splash

Name, Info

Stats

For each Spell:

Spell pic link

Spell name, description

Spell stat numbers, etc

If –t:

For each Tip:

Tip

If –l:

Lore (broken up into fragments because of discord’s word limit)

If –s:

Show skin links

Link to lol wiki and lol official

* + - * Lore – Get champion lore
        + Usage: \lore name
        + Needs:

Datadragon

* + - * + Display:

Broken up lore fragments

* + - * Skins – Get champion skins
        + Usage: \skins name –r=region –p=patch
        + Needs:

Datadragon url for champion skin art

* + - * + Display:

For each art:

Link to art

* + - * Icon – Get profile icons
        + Usage \icon id|-r

-r: random id

* + - * + Needs:

Datadragon url for icon art

* + - * + Display:

Link to icon

* + - * Emote – Get emotes
        + Usage \emote id|-r
        + Needs:

Datadragon url for emote art

* + - * + Display:

Link to emote

* + - * Item – Get item info
        + Usage: \item name –p=patch
        + Needs:

Datadragon url for items

* + - * + Display:

Item art link

Item name, cost

Stats and Effects

Builds into:

* + - * Best – Get top k best champions in a role
        + Usage: \best role –k=number
        + Needs:

Champion.gg

* + - * + Display:

For each champion:

Name, win, play, ban

* + - * Stats – Get stats for a champion
        + Usage: \stats name –r=region –e=rank –l=role –p=patch
        + Needs:

Champion.gg

* + - * + Display:

Name, Role, Ranks, Patch, Region

Averaged from X games

Win, play, ban, role

Stats: kda, cs, monsters, damage, healing

Rankings compared to others:

For each Score:

Score: current up/down diff from previous

Max/Min Scores:

For each Score:

Score: Min to Max

* + - * Build – Get build for a champion
        + Usage: \build name –l=role
        + Needs:

Champion.gg

* + - * + Display:

Name, role

For each Build:

For each sequence:

Item sequence ->

* + Managers
    - File – Downloads files from urls and saves them.
      * Downloads file from the given url to file path.
      * Checks if file already exists before downloading.
      * Resizes images if needed after downloading.
    - Database – Interfaces with static database.
      * Connects with database to retrieve static info.
      * For dev purposes only, will have methods for inserting into database and resetting database.
    - Cache – Temporarily holds data in memory to prevent API overload.
      * Holds Database info in cache to minimize database calls.
      * Holds API info in cache to minimize API calls.
    - Creator – Creates data structures from other data structures
      * Create specialized data structures from stored API results and Database calls.
  + Data
* Problems with Display of Data Structures
  + Discord Chat Message Character Limit – Forces string into section breaks
  + Some data structures need a level of detail so it knows what info to hide. Cant easily use \_\_str\_\_
  + Data Structures have lots of info, so their constructors are big and messy
  + Tabs need to go one more than before when going deeper
* Solution?
  + Every Data Struct has a to\_str(detail(optional), depth)
  + This returns a list of strings for the calling command method to iterate over and say.
  + Detail helps the command method control how much or what info to show
  + Depth helps with the tab indentations
  + Constructors will still be fairly large and messy, but this is a product of having pure data structures that holds lots of info.
  + Use pairs, tuples, lists, and smaller structs when you can
* For matches
  + Need summoner spell db maps
  + Need rune db maps + var names
  + Need item db maps
  + Need a has\_lane and has\_score var
  + Need to add has\_objs & has\_scores to queues db
  + Add extra \n in between major sections
* Refactoring + Design
  + Disco Bot is essentially a collection of Cogs and Managers.
    - Each Cog has a suite of Commands it hosts.
    - Each Manager controls one utility aspect to help the Cogs.
  + LoL Cog
    - Each Command follows a generic series of execution:
      * Parse Input into Inputs and Arguments
      * Parse specific Inputs and Arguments from Generic Ones.
      * Check Input Legitimacy
      * Check Cache with Key
      * If Cache exists:
        + Display Cache
      * If Cache does not exist:
        + Get Data Objects via API Calls.

Check API Cache first.

Use the Cache instead if it exists

This checks if the inputs are good as well

This can possibly make Database calls.

* + - * + Create Data Structure Container via Data Factory.

This will create the structure from the data objects and inputs.

This can possibly make Database calls.

* + - * + Cache the structure.
        + Display the object.
    - Packages are small data structures that are used with the larger, core data structures.
      * Packages are used to make data structure construction easier.
    - Factory Methods are methods that create the data structure from data objects and inputs. Each command has an associated factory method. Factory methods can have helper factory methods.
      * Factory methods assume the parameters given are valid.
      * Factory methods are all different, but share this sequence:
        + Get any database info from database.
        + Create packages from helper factories.
        + Put together to create the data structure.
  + Data Structures
    - All data structures and packages have a to\_str(depth) method that creates a list of strings for the caller to iterate over and display.
    - Data Structures have a constructor.
  + Cache Manager
    - Each public method is a get/set for each cache.
    - Get Cache
      * Gets the cached object associated with the key if it exists and is fresh.
      * Returns None if key does not exist, or is not fresh.
      * If the object is returned, its freshness is refreshed.
    - Add Cache
      * Adds the object associated with the key.
      * Overwrites if the key exists already.
    - Clean Up/Free
      * Cleans up un-fresh cache objects by removing them and free memory
    - Check if Exists
      * Private method
      * Checks if the key exists in each cache.
    - Check if Fresh
      * Private method
      * Checks if the object associated with the key is fresh.
  + Database Manager
    - Each public method is a select query for a specific data type.
    - Each select will generally follow this:
      * Check input validity.
      * Check cache.
      * If cache exists:
        + Return the cached result in a fancy tuple/list
      * If cache does not exist:
        + Make database call.
        + Cache the result.
        + Return the result in a fancy tuple/list.
      * Can return None if the query returns None, or if the inputs are wrong.
  + File Manager
    - A simple collection of file-related methods.
    - Download File
    - Check File
* Cogs
  + DiscoGeneralCog
  + DiscoLoLCog
* Commands
  + Player
    - Input Parsing
      * Player Name :r=region
      * Player requires a single input: Name.
      * Player also has a single argument: Region.
      * Helper Method:
        + Parse into Input and Args (Input String)
        + Extract Region from Args (Args, r)

If Region is not given or None, use the DEFAULT REGION

* + - Input Validation
      * No restrictions on Name or Region.
    - API Calls
      * Player needs two API calls: Summoner, and Ranks.
      * Helpers:
        + Get Summoner From Name and Region (name, region)
        + Get Rank From Id and Region (id, region)
    - Structure Factory
      * Player needs a Player structure.
      * Helpers:
        + Create Player From Summoner and Rank (Summoner, Rank)
        + Create Rank From Rank (Rank)
    - Display
  + Matchlist
  + Match
  + Timeline
  + Commands hold Get API methods, Parse and Extract methods, Cache, and Display methods.
* Managers
  + Database
  + Cache
  + File
* Structures
  + Primary data structures and any helper structures.
  + Also holds factory methods for the structures
  + Structures are comprised of smaller structure components.
  + Each component takes a few arguments to construct.
* Data
  + Any downloaded data-
  + Images, JSON, database
* Values – Constants and Enumerations
  + Lol Values
  + General Values
  + Enumerations are stored in Values with Constants
  + Mappings are stored in Values with Constants