vTR census library

1. Downloaded fasta and tab file for proteins identified in vTR census (list of UniProtKB IDs from supplemental table in Liu et al. 2020 paper).
2. Unzipped with gunzip
3. Used uniprotkb_fasta2csv.py to compile representative sequence and metadata into csv.
4. Manually curated a list of BSL4 viruses and used this to filter out BSL4 virus proteins from the vTR census with remove_BSL4_viruses.py.
5. Used generate_tiles_v2.py to make tiles (size: 80AA, window: 10AA), removing duplicates.
6. Ran domains_to_codon_opt_oligos_v2.py to generate codon-optimized DNA sequences from the tile protein sequences; default maximum GC content to enforce is 65%, but if codon optimization fails, then I relax the constraint by +1%, try again, and so on (this is to meet Twist's upper limit of 65% as best as possible).
7. Performed QC on codon-optimized oligos, specifically looking at population-level codon usage with qc_oligos_codon_usage.py and the distribution of oligo GC content with qc_oligos_GC_content.py. I also opened the csv and searched for BsmBI sites (CGTCTC) and C homopolymers equal to or greater than 7 in length.

Human herpesvirus library

1. Downloaded fasta and tab file for human herpesviruses (host human, 90% identity, reviewed, and specifically exclude molloscum contagiosum which is included otherwise for some reason) and for pseudorabies virus (SuHV, 90% identity, reviewed). Placed in separate folders.
    a. Search term: uniprot:(herpesvirus host:human NOT molluscum reviewed:yes) AND identity:0.9
2. Unzipped with gunzip.
3. For each separate set of fasta+tab, used uniref_fasta2csv.py to compile representative sequence and metadata into csv. For human herpesviruses, two proteins contained at least one X, so script separated data into two files, one for proteins lacking X and the other for proteins containing X. For SuHV, there were no Xs, so all proteins in one file.
4. Manually inspected/corrected X-containing human herpesvirus proteins in Excel and saved as a new csv. Combined this csv with the human herpesvirus one for proteins lacking X and with the SuHV csv using compile_dataframes.py.
5. Used generate_tiles_v2.py to make tiles (size: 80AA, window: 10AA) of all proteins in HHV+SuHV csv, removing duplicates.
6. Ran domains_to_codon_opt_oligos_v2.py to generate codon-optimized DNA sequences from the tile protein sequences; default maximum GC content to enforce is 65%, but if codon optimization fails, then I relax the constraint by +1%, try again, and so on (this is to meet Twist's upper limit of 65% as best as possible).
7. Performed QC on codon-optimized oligos, specifically looking at population-level codon usage with qc_oligos_codon_usage.py and the distribution of oligo GC content with qc_oligos_GC_content.py. I also open the csv and searched for BsmBI sites (CGTCTC) and C homopolymers equal to or greater than 7 in length.

8. Mapped virus to subfamily and subdivided library by subfamily (alpha, beta, gamma) with generate_herpesvirus_sublibraries.py.

Coronavirus library

1. Determined which coronaviruses to study (11 in total) via literature review.
2. Downloaded fasta and tab files for their proteins in UniProtKB (most reviewed, some not), specifically excluding a Marburg virus protein and human protein that appeared in the more general search.
3. Unzipped with gunzip.
4. For each separate set of fasta+tab, used uniprotkb_fasta2csv.py to compile representative sequence and metadata into csv.
5. Because a number of entries were large ORF1ab replicase polyproteins, used polyprotein_generate_IDs.py to compile these entries into a list (txt file).
6. Uploaded this file to UniProt's 'Retrieve/ID mapping' tool, edited the columns shown to include 'Chain' under 'PTM/Processing', and downloaded the fasta and tab file associated with this list.
7. Unzipped with gunzip.
8. Used uniprotkb_fasta2csv.py to compile representative sequence and metadata (including chain info this time) into csv.
9. Inspected csv and found one polyprotein (BtCoV RaTG13 ORF1ab) missing chain info (not a reviewed protein). Because of its very high sequence identity to SARS-CoV-2 ORF1ab, copied SARS-CoV-2 annotations with minor adjustment (accounted for insertion of isoleucine at 1023).
10. Used polyprotein2chains.py to process polyprotein into annotated proteolytic fragments.
11. Used generate_tiles_v2.py on the sets of polyprotein- and non-polyprotein-derived proteins (in separate files at this point) to make tiles (size: 80AA, window: 10AA), removing duplicates.
12. Combined these csv files with compile_dataframes.py.
13. Ran domains_to_codon_opt_oligos_v2.py to generate codon-optimized DNA sequences from the tile protein sequences; default maximum GC content to enforce is 65%, but if codon optimization fails, then I relax the constraint by +1%, try again, and so on (this is to meet Twist's upper limit of 65% as best as possible).
14. Performed QC on codon-optimized oligos, specifically looking at population-level codon usage with qc_oligos_codon_usage.py and the distribution of oligo GC content with qc_oligos_GC_content.py. I also opened the csv and searched for BsmBI sites (CGTCTC) and C homopolymers equal to or greater than 7 in length.

Immune controls library

1. Did research to find proteins and domains involved in a number of immune-related processes. Compiled a list of UniProt IDs and uploaded to the 'Retrieve/ID mapping' tool.
2. Downloaded fasta and tab file for these proteins.
3. Unzipped with gunzip.
4. Used uniprotkb_fasta2csv.py to compile representative sequence and metadata into csv.
5. Manually extracted domains and mutagenized proteins in Excel where applicable.

6. Used generate_tiles_v2.py to make tiles (size: 80AA, window: 10AA), removing duplicates.
7. Ran domains_to_codon_opt_oligos_v2.py to generate codon-optimized DNA sequences from the tile protein sequences; default maximum GC content to enforce is 65%, but if codon optimization fails, then I relax the constraint by +1%, try again, and so on (this is to meet Twist's upper limit of 65% as best as possible).
8. Performed QC on codon-optimized oligos, specifically looking at population-level codon usage with qc_oligos_codon_usage.py and the distribution of oligo GC content with qc_oligos_GC_content.py. I also opened the csv and searched for BsmBI sites (CGTCTC) and C homopolymers equal to or greater than 7 in length.

Generating random controls

1. Edited Nicole's generateRandomers.py script to use my DNA chisel codon optimization parameters.
2. Used this script to generate 500 randomers that were 240nt in length.
3. Performed QC on codon-optimized oligos, specifically looking at population-level codon usage with qc_oligos_codon_usage.py and the distribution of oligo GC content with qc_oligos_GC_content.py. I also opened the csv and searched for BsmBI sites (CGTCTC) and C homopolymers equal to or greater than 7 in length.