# Graduate School Class Reminders

- Maintain six feet of distancing
- Please sit in the same chair each class time
- Observe entry/exit doors as marked
- Use hand sanitizer when you enter/exit the classroom
- Use a disinfectant wipe/spray to wipe down your learning space before and after class
- Media Services: 414 955-4357 option 2

# Documentation on the web

- CRAN: `http://cran.r-project.org`
- R manuals: `https://cran.r-project.org/manuals.html`
- SAS: `http://support.sas.com/documentation`
- SAS 9.3: `https://support.sas.com/en/documentation/documentation-for-SAS-93-and-earlier.html`
- Step-by-Step Programming with Base SAS 9.4 (SbS): `https://documentation.sas.com/api/docsets/basess/9.4/content/basess.pdf`
- SAS 9.4 Programmer s Guide: Essentials (PGE): `https://documentation.sas.com/api/docsets/lepg/9.4/content/lepg.pdf`
- Wiki: `https://wiki.biostat.mcw.edu` (MCW/VPN)

# DATASTEP: arrays

- Arrays are a temporary construct of the DATASTEP
  Their component variables may, or may not, be saved in the `NEW` data set, but the array's definition is NOT
- Numeric: `array _VAR(n) VAR1 ... VARn;`
  `_VAR(n)` can be an asterisk: `_VAR(*)`
- Character: `array _CHAR(m) $ L CHAR1 ... CHARm;`
- `array _VAR(n:m) VARn ... VARm;`
- Temporary: variables are not named and not saved
  `array _VAR(n) _TEMPORARY_;`
  `_VAR(n)` can NOT be an asterisk
- Optionally, with initial values assigned
  `array _VAR(n) _TEMPORARY_ (VALUE1 ... VALUEn);`
- Two dimension arrays: row major
  `array _VAR(n, m) VAR1 ... VARnm;`
  `_VAR(n, m)` can NOT be an asterisk
- Multiple dimension arrays
  `array _VAR(n, ..., m) ...;`

# DATASTEP: arrays

- You reference array values like so
- For `array _VAR(n) VAR1 ... VARn;`
  `_VAR(i)`
- However, that is ambiguous since now SAS will NOT call the function `_VAR()`
  a reason that I like to use the underscore in array names since there are no such functions
  (YET: we will re-visit that next lecture)
- But, for most flexibility
  (such as in the `PROC FCMP` which we will see next time)
  use `_VAR[i]`

# DATASTEP: iterative DO loops

- The `do` loop has a few variants
- `do VAR=VALUE1 to VALUE2 by VALUE3; ...; end;`
  if `VALUE3` not given, it defaults to 1
- `do VAR=VALUE1 to VALUE2 until(CONDITION); ...; end;`
  increments until the `CONDITION` is true or `VALUE2` whichever comes first
- `do VAR=VALUE1 to VALUE2 while(CONDITION); ...; end;`
  increments until the `CONDITION` is false or `VALUE2` whichever comes first

# SAS formats

- In Chapters 4 and 5, we saw informats for inputting data
- For output, formats end in a period: `FORMAT.`
  with an optional WIDTH `FORMATw.` potentially along with a number of DECIMAL places `FORMATw.d`
- Common numeric formats:
  `data NEW; set OLD; format X best12. Y 6.4 Z z4.;`
- `Zw.` and `Zw.d` are filled with leading zeros
- Common character formats: they start with dollar sign
  `data NEW; set OLD; format X $40. Y $char200.;`
- SAS supplies many formats and informats
- You can make your own with `proc format`

# SAS dates

- There are other data types besides numeric and character
  the most common is a SAS date
  with literals that look like this
  `"DDMONYYYY"d` GOOD
  `"DDMONYY"d` BAD
  due to Y2K, and other types of 2-digit year errors,
  the former is far preferable to the latter (see below)
- For example, `"12OCT2020"d` which is the numeric value
  22200: the number of days from the SAS origin 01/01/1960
  (which is the typical choice for IBM mainframes
  as opposed to 01/01/1970 of Unix/R)
- They are NOT ISO 8601 compliant: dates prior to 1582
  generate an error
- And a Y2K-like bug/feature
  `"12OCT0000"d` is equivalent to `"12OCT2000"d`
  `"0000"` is considered to be the two-digit year `"00"` or `"2000"`
  this behavior is documented so it is a *feature*
  (SAS is pretty good about fixing bugs due to their rental fees)

# DATASTEP: SAS dates

- There are many useful SAS features for SAS dates
- Such as the SAS format `date9.`
- And the automatic macro variable `sysdate9` which is formatted as `date9.`
- There are several useful SAS data functions
- For example, `intnx` and `intck`
- `intnx` creates a date offset into the future or the past
- `intck` counts the number of intervals between two dates useful for calculating age
- see `lecture13.sas`
- There are also times `"HH:MM:SS"t`
- And datetimes `"DDMONYYYY:HH:MM:SS"dt`
- But these are uncommon except for EMR/clinical trials

# DATASTEP: SELECT statement type 1

RED and BLUE are optional

```
select(EXPRESSION);
when(VALUE1) BLOCK1
when(VALUE2) BLOCK2
    ⋮
when(VALUEn) BLOCKn
otherwise BLOCKO
end;
```

- if VALUE1, ..., VALUEn are not exhaustive
  AND there is no otherwise clause,
  then errors are generated for those observations that remain

# DATASTEP: SELECT statement type 2

RED and BLUE are optional

```
select;
when(CONDITION1) BLOCK1
when(CONDITION2) BLOCK2
    ⋮
when(CONDITIONn) BLOCKn
otherwise BLOCK0
end;
```

- if CONDITION1, ..., CONDITIONn are not exhaustive
  AND there is no otherwise clause,
  then errors are generated for those observations that remain

# HW part 1: create a date format for the ISO 8601 Proleptic Gregorian Calendar

- The details are in lecture 2, slides 3-5
- See my example of the Proleptic Julian Calendar `julian.sas`

# HW part 2: NTDB hot-decking

- Write a SAS DATASTEP program to perform hot-decking for the NTDB example: see the details in lecture 4, slide 4
- Hints: for random number generation from the Uniform distribution, use the `rand("unif")` and `call streaminit(SEED);` to set the seed.
- Use arrays with the `array` statement
- Use the `set` statement with the `point` option to iterate through the data set BEWARE: with `point`, don't forget the `stop; run;` at the end of the DATASTEP
- With many functions you can use an `of` clause with a variable list, e.g., `nmiss(of VAR1-VARn)` instead of `nmiss(VAR1, ..., VARn)`
- Or with the STARTS-WITH colon operator `nmiss(of VAR:)`
- Use the `%_nobs()` macro to determine how many observations are in a data set like `%_nobs(data=ntdb.elder)`