Master of Science in
Genomics and Bioinformatics

Lectures 8
OOP

GNBF5010

Instructor: Jessie Y. Huang

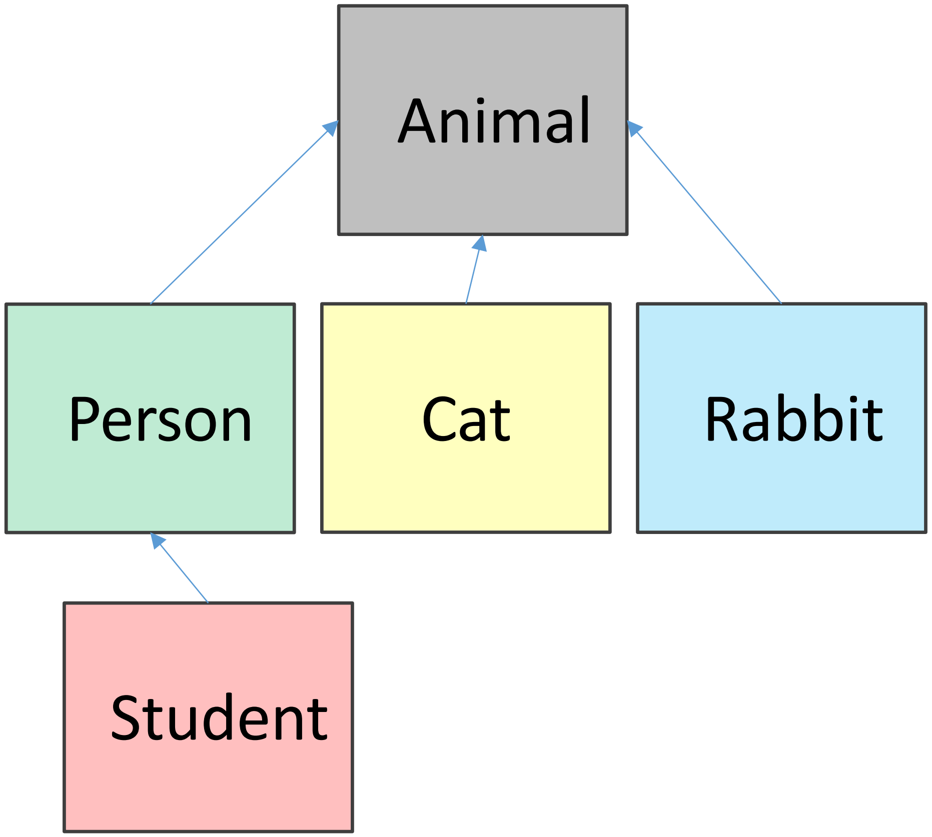
Overview

- Inheritance
- OOP Example: Counting SNPs

INHERITANCE

Inheritance and the "is-a" relationship

- Parent class (superclass)
- Child class (subclass)
  - Inherits all data and behaviors of parent class
  - Can add more info
  - Can add more behavior
  - Can override behavior



Animal: The superclass

```
class Animal:
    def __init__ (self, age):
        self.age = age
        self.name = ""
    def get_age (self):
        return self.age
    def get_name (self):
        return self.name
    def set_age (self, newage):
        self.age = newage
    def set_name (self, newname=""):
        self.name = newname
    def __str__ (self):
        return f'Animal: {self.name}:{self.age}'
```

Cat: A subclass of Animal

Inherit all attributes and methods of the Animal class

```
class Cat(Animal):
    def speak(self):
        print("meow")
    def __str__(self):
        return f"Cat: {self.name}:{self.age}"
```

Add new method of speak()

Override the __str__() method of Animal

- Add new functionality with speak()
  - Instance of type `Cat` can call this new methods
  - Instance of type `Animal` throws error if it calls Cat's new method
- `__init__()` is not missing, it just uses the Animal version

Which method to use?

- Subclass can have methods with the same name as superclass.
- An instance of a class will look for the method name in current class definition.
- If not found, look for the method name up the hierarchy (in parent, then grandparent, and so on).
- Use the first method found in the hierarchy with the method name, which means a parent's method can be overridden.

Person: Another subclass of Animal

```
class Person(Animal):
    def __init__(self, name, age):
        Animal.__init__(self, age)
        self.set_name(name)
        self.friends = []
    def get_friends(self):
        return self.friends
    def add_friends(self, fname):
        if fname not in self.friends:
            self.friends.append(fname)
    def speak(self):
        print("Hello!")
    def age_diff(self, other):
        diff = self.age - other.age
        print(abs(diff), "years difference")
    def __str__(self):
        return f"Person: {self.name}:{self.age}"
```

Call Animal's constructor, set an attribute add a new data attribute.

New methods

Override the __str__() method of Animal

Student: A subclass of Person

```
class Student(Person):
    def __init__(self, name, age, major=None):
        Person.__init__(self, name, age)
        self.major = major
    def change_major(self, major):
        self.major = major
    def speak(self):
        r = random.random() # return float in [0,1]
        if r < 0.25:
            print("I have homework.")
        elif 0.25 <= r < 0.5:
            print("I need sleep.")
        elif 0.5 <= r < 0.75:
            print("I should eat.")
        else:
            print("i am watching tv.")
    def __str__(self):
        return f"Student:{self.name}:{self.age}:{self.major}"
```

Student inherits both Person and Animal's attributes and methods.

The main() function

```python
def main():
    animalA = Animal(2)
    animalA.set_name("Gigi")
    print(str(animalA))

    catA = Cat(1)
    catA.set_name("HelloKitty")
    catA.speak()
    print(str(catA))

    personA = Person("Ann", 20)
    personA.add_friends("Leo")
    personA.add_friends("Biff")
    print(personA.get_friends())

    personB = Person("Zack", 25)
    personB.age_diff(personA)

    studentA = Student("Grace", 19, "Finance")
    studentA.speak()
    studentA.speak()
    print(str(studentA))
```

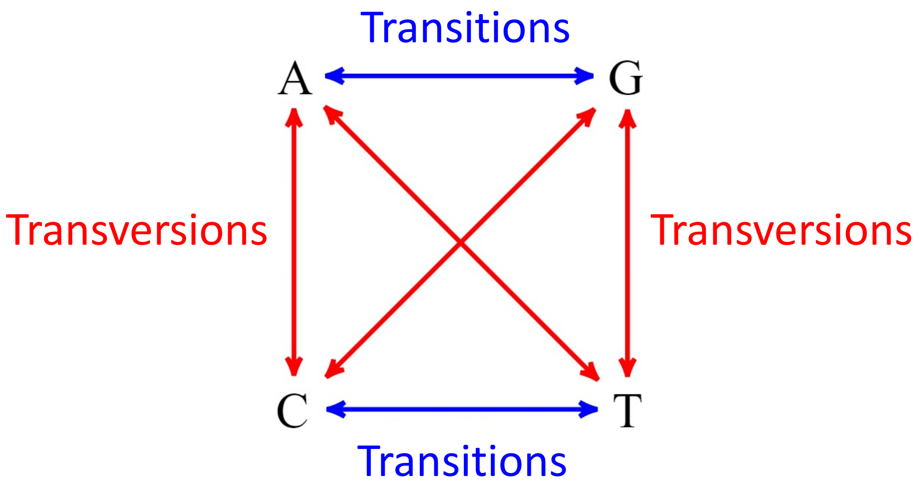
OOP Example: Counting SNPs

Question 1

- The file **trio.sample.vcf** represents a random sampling of **SNPs** from three people (a mother, a father, and their daughter) compared to the reference human genome.

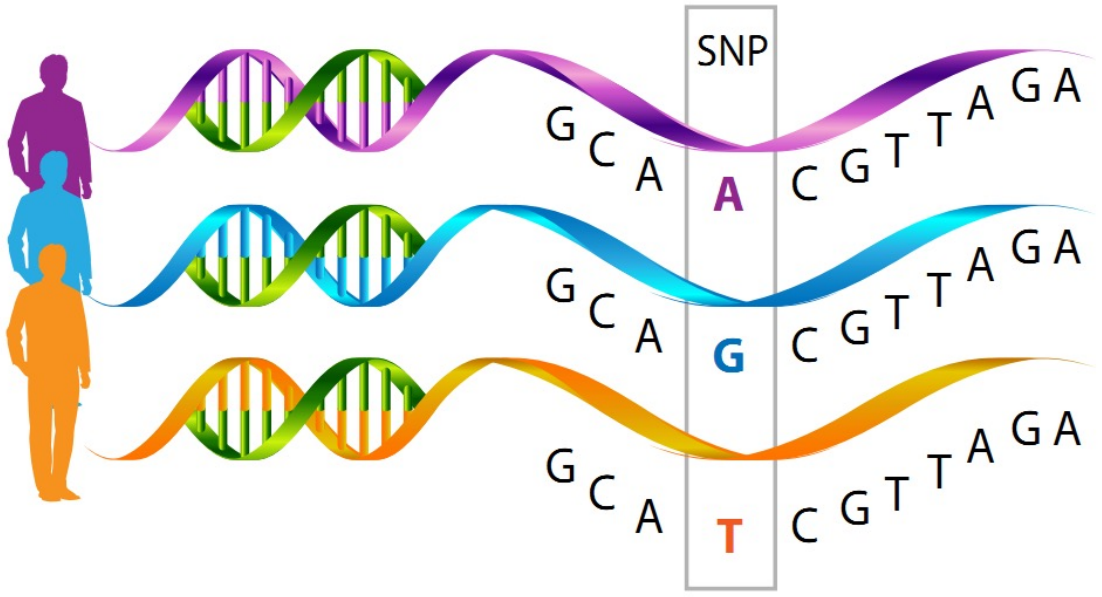
a. How many **transition SNPs** (A vs. G or C vs. T) are there within **each chromosome**?

b. How many **transversion SNPs** (anything else) are there within **each chromosome**?



What is a SNP?

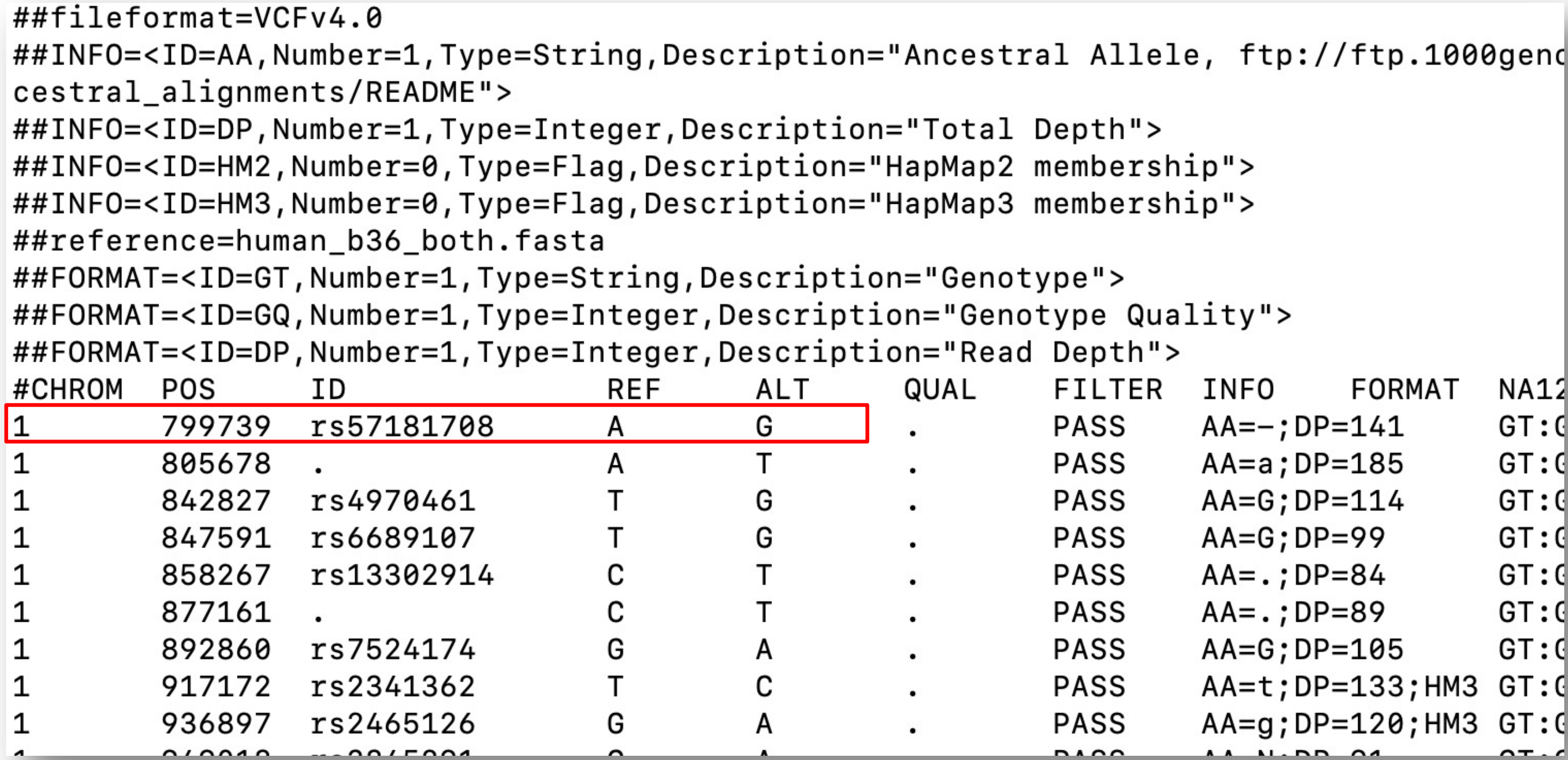
- SNP (pronounced 'snip') stands for **Single-Nucleotide Polymorphism.**
- A genetics term for a site in DNA, which varies within a population.
- Many SNPs affect different traits, and thus can be used to predict traits or disease risk!



The VCF (Variant Call Format) file

- For storing sequence variants like SNPs.

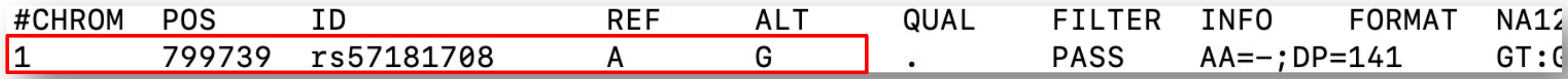
<u>`trio.sample.vcf`</u>



```
##fileformat=VCFv4.0
##INFO=<ID=AA,Number=1,Type=String,Description="Ancestral Allele, ftp://ftp.1000genc
cestral_alignments/README">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Total Depth">
##INFO=<ID=HM2,Number=0,Type=Flag,Description="HapMap2 membership">
##INFO=<ID=HM3,Number=0,Type=Flag,Description="HapMap3 membership">
##reference=human_b36_both.fasta
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
#CHROM  POS     ID              REF     ALT     QUAL    FILTER  INFO            FORMAT  NA12
1       799739  rs57181708      A       G       .       PASS    AA=-;DP=141             GT:G
1       805678  .               A       T       .       PASS    AA=a;DP=185             GT:G
1       842827  rs4970461       T       G       .       PASS    AA=G;DP=114             GT:G
1       847591  rs6689107       T       G       .       PASS    AA=G;DP=99              GT:G
1       858267  rs13302914      C       T       .       PASS    AA=.;DP=84              GT:G
1       877161  .               C       T       .       PASS    AA=.;DP=89              GT:G
1       892860  rs7524174       G       A       .       PASS    AA=G;DP=105             GT:G
1       917172  rs2341362       T       C       .       PASS    AA=t;DP=133;HM3 GT:G
1       936897  rs2465126       G       A       .       PASS    AA=g;DP=120;HM3 GT:G
```

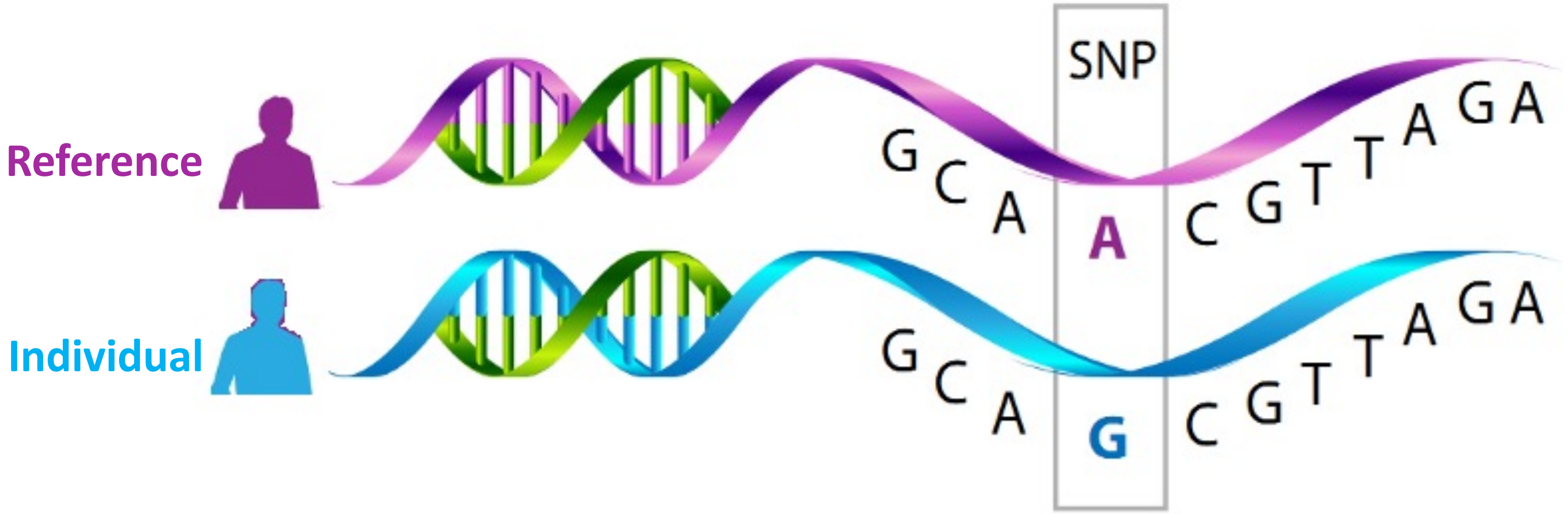
The VCF (Variant Call Format) file

- For example, the first SNP in `trio.sample.vcf`



```
#CHROM  POS     ID          REF   ALT   QUAL  FILTER  INFO          FORMAT  NA12
1       799739  rs57181708  A     G     .     PASS    AA=-;DP=141           GT:G
```

is corresponding to the following SNP:



- Located at Position 799739 of Chromosome 1
- The reference allele is A, alternative allele is G
- This SNP is a transition (A<->G)

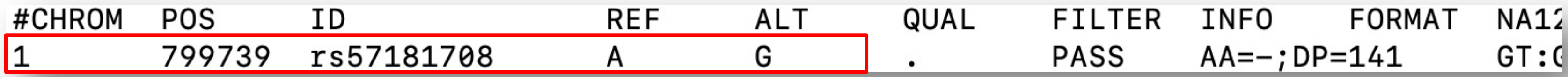
Overall strategy

[Note]: We don’t have to use OOP to counting transitions and transversions. But OOP can help answer related questions about the same data easily.

- First of all, define SNP class and Chromosome class
- Next, create a collection of chromosome objects that we can add SNP objects to.
  - Better to keep chromosome objects in a dictionary,
    - with chromosome name as the key and
    - chromosome object as the value.
- Then, loop through chromosome objects and ask each how many transitions and transversions it has.

Design the SNP class

- A SNP object will hold relevant information about a single line in the VCF file
- **Attributes:**
  - The first five columns in the VCF
  - All values be provided in the **constructor**
- **Methods:**
  - `is_transition()`
  - `is_transverstion()`



```
#CHROM  POS     ID          REF   ALT   QUAL   FILTER  INFO         FORMAT  NA12
1       799739  rs57181708  A     G     .      PASS    AA=-;DP=141          GT:
```

The SNP class

```
# A class representing simple SNPs
class SNP:
    def __init__(self, chrname, pos, snpid, ref_allele, alt_allele):
        assert ref_allele != alt_allele, f"Error: ref == alt at pos {pos}"
        self.chrname = chrname
        self.pos = pos
        self.snpid = snpid
        self.ref_allele = ref_allele
        self.alt_allele = alt_allele

    # Returns True if ref_allele/alt_allele is A/G, G/A, C/T, or T/C
    def is_transition(self):
        is_AG = (self.ref_allele == "A" and self.alt_allele == "G")
        is_GA = (self.ref_allele == "G" and self.alt_allele == "A")
        if is_AG or is_GA:
            return True

        is_CT = (self.ref_allele == "C" and self.alt_allele == "T")
        is_TC = (self.ref_allele == "T" and self.alt_allele == "C")
        if is_CT or is_TC:
            return True

        return False

    # Returns True if the snp is a transversion (ie, not a transition)
    def is_transversion(self):
        if self.is_transition():
            return False
        return True
```

The **`assert`** statement checks the condition (first argument). If it's not true, the program will be aborted and report the error message (second argument).

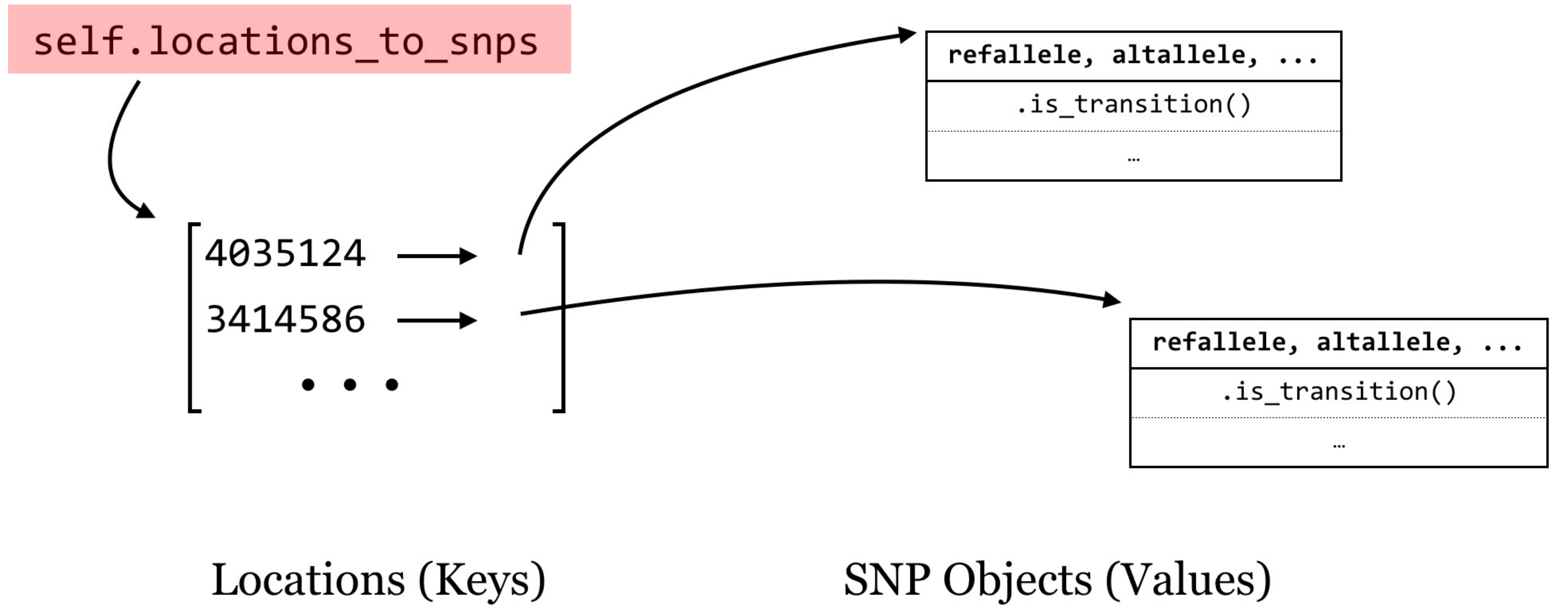
The SNP class

```
    # For nice print
    def __str__(self):
        return f"chrname = {self.chrname}\n" + \
               f"pos = {self.pos}\n" + \
               f"snpid = {self.snpid}\n" + \
               f"ref = {self.ref_allele}\n" + \
               f"alt_allele = {self.alt_allele}\n" + \
               f"is_transition = {self.is_transition()}\n" + \
               f"is_transversion = {self.is_transversion()}\n"


# Transition test; should not result in "Test failed!"
snp1 = SNP("1", 12351, "rs11345", "C", "T")
assert snp1.is_transition(), "Test failed!"
print(snp1)
print()

# Transversion test; should not result in "Test failed!"
snp2 = SNP("1", 36642, "rs22541", "A", "T")
assert snp2.is_transversion(), "Test failed!"
print(snp2)
print()

# Error test; should result in "Error: ref == alt at position 69835"
snp3 = SNP("1", 69835, "rs53461", "A", "A")    # Program aborted here
```

For debug only; will be commented out later

Design the Chromosome class: Attributes

- **chrname**: chromosome name, e.g. ‘1’, ‘2’, ‘x’
- **location_to_snp:** A **dictionary** of SNPs objects that are located on the chromosome, with location as the key



Design the Chromosome class: Methods

- `__init__()`
  - Initialize the name of the chromosome as **`self.chrname`**
  - Initialize **an empty dictionary** for the SNPs, which will be revised later
- `count_transitions()`
  - Returns the number of transition SNPs
- `count_transversions()`
  - Returns the number of transversion SNPs
- `add_snp()`
  - Given the information of a SNP, create a SNP object, and add it to the SNP dictionary, location_to_snp.

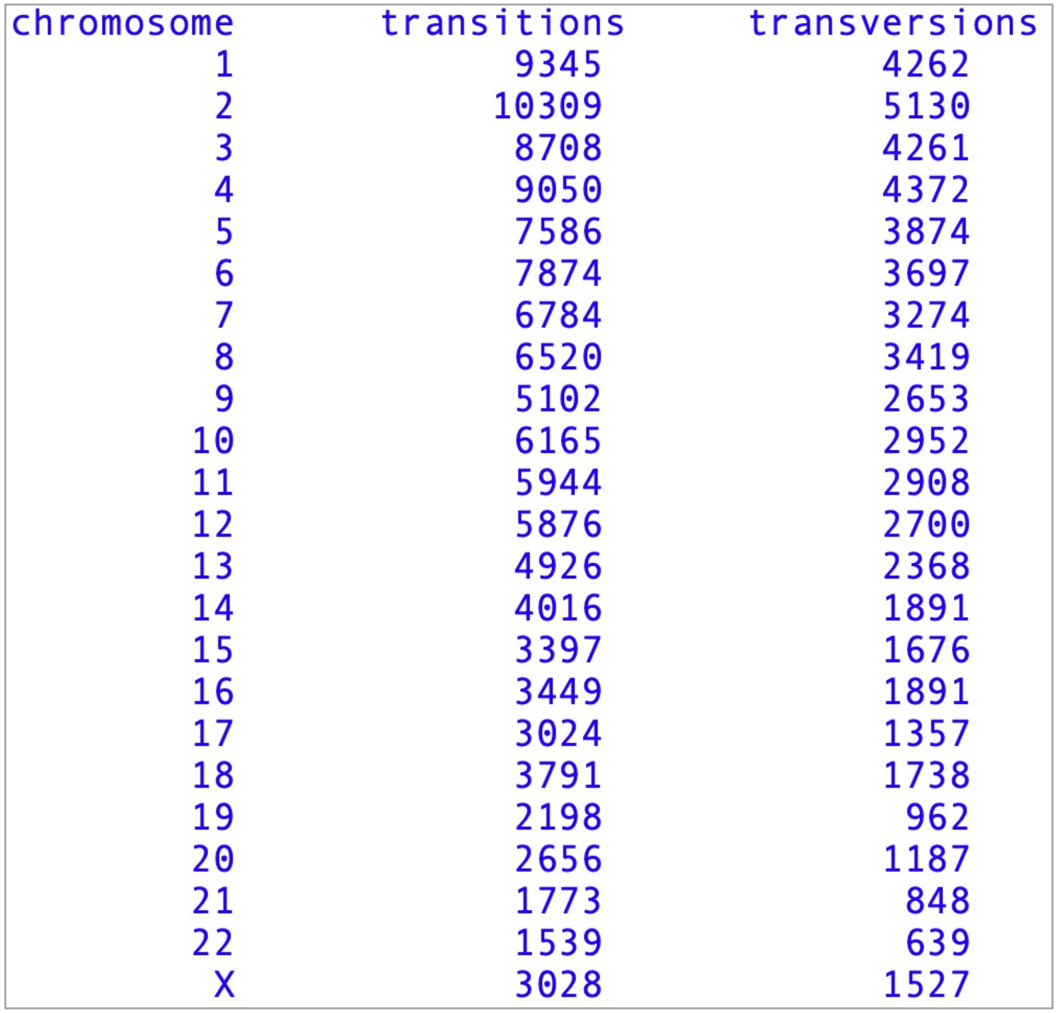
The Chromosome class

```
# A class representing a chromosome, which has a collection of SNPs
class Chromosome:
    def __init__(self, chrname):
        self.chrname = chrname
        self.locations_to_snps = dict()

    def get_name(self):
        return self.name

    # Given all necessary information to add a new SNP, create
    # a new SNP object and add it to the SNPs dictionary.
    def add_snp(self, chrname, pos, snpid, ref_allele, alt_allele):
        newsnp = SNP(chrname, pos, snpid, ref_allele, alt_allele)
        self.locations_to_snps[pos] = newsnp

    # Returns the number of transition snps stored in this chromosome
    def count_transitions(self):
        count = 0
        for snp in self.locations_to_snps.values():
            if snp.is_transition():
                count = count + 1
        return count

    # Returns the number of transversion snps stored in this chromosome
    def count_transversions(self):
        return len(self.locations_to_snps) - self.count_transitions()


# Test chromosome class
chr1 = Chromosome("testChr")
chr1.add_snp("testChr", 24524, "rs15926", "G", "T")
chr1.add_snp("testChr", 62464, "rs61532", "C", "T")

# These should not fail:
assert chr1.count_transitions() == 1, "Test Failed!"
assert chr1.count_transversions() == 1, "Test Failed!"
```

The `main()` function

```python
def main():

    # Create chrnames_to_chrs dictionary, parse the input file
    chrnames_to_chrs = dict()
    filename = "trio.sample.vcf"
    with open(filename, "r") as fh:
        for line in fh:
            # Skip header lines, which starts with #
            if not line.startswith("#"):
                fields = line.strip().split("\t")
                chrname = fields[0]
                pos = int(fields[1])
                snpid = fields[2]
                ref = fields[3]
                alt = fields[4]

                # Load the data into the dictionary
                if chrname not in chrnames_to_chrs:
                    chrnames_to_chrs[chrname] = Chromosome(chrname)
                chrnames_to_chrs[chrname].add_snp(chrname, pos, snpid, ref, alt)

    # Print the results!
    print("chromosome\t" + "transitions\t" + "transversions")
    for chrname in chrnames_to_chrs:
        chr_obj = chrnames_to_chrs[chrname]
        trs = chr_obj.count_transitions()
        trv = chr_obj.count_transversions()
        print(f"{chrname:>10s}\t{trs:10d}\t{trv:10d}")
```

Program output

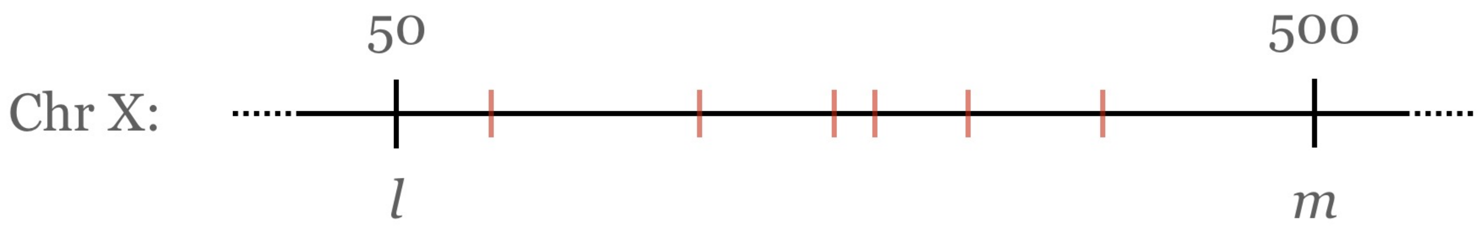


```
chromosome        transitions      transversions
         1               9345               4262
         2              10309               5130
         3               8708               4261
         4               9050               4372
         5               7586               3874
         6               7874               3697
         7               6784               3274
         8               6520               3419
         9               5102               2653
        10               6165               2952
        11               5944               2908
        12               5876               2700
        13               4926               2368
        14               4016               1891
        15               3397               1676
        16               3449               1891
        17               3024               1357
        18               3791               1738
        19               2198                962
        20               2656               1187
        21               1773                848
        22               1539                639
         X               3028               1527
```

Question 2 (an extension):
Determine the most SNP-dense region of each chromosome

Calculate the SNP density

- Given a region from positions *l* to *m*, the density is
  - the number of SNPs occurring within *l* and *m* divided by
  - the size of the region *(m – l + 1),* times
  - 1,000 (for SNPs per 1,000 base pairs).



$$\text{SNPs per 1000 base pairs} = \frac{6}{500-50+1} \times 1000 \approx 13.3 \text{ SNPs per 1Kb}$$

How to scan the chromosome?

- If region size = 100,000 bps, then consider
- bases 1 to 100,000 to be a region,
- 100,001 to 200,000 to be a region,
- and so on,
- up until the start of the region considered exceeds the last SNP location.

- This can be accomplished with a **while-loop**.

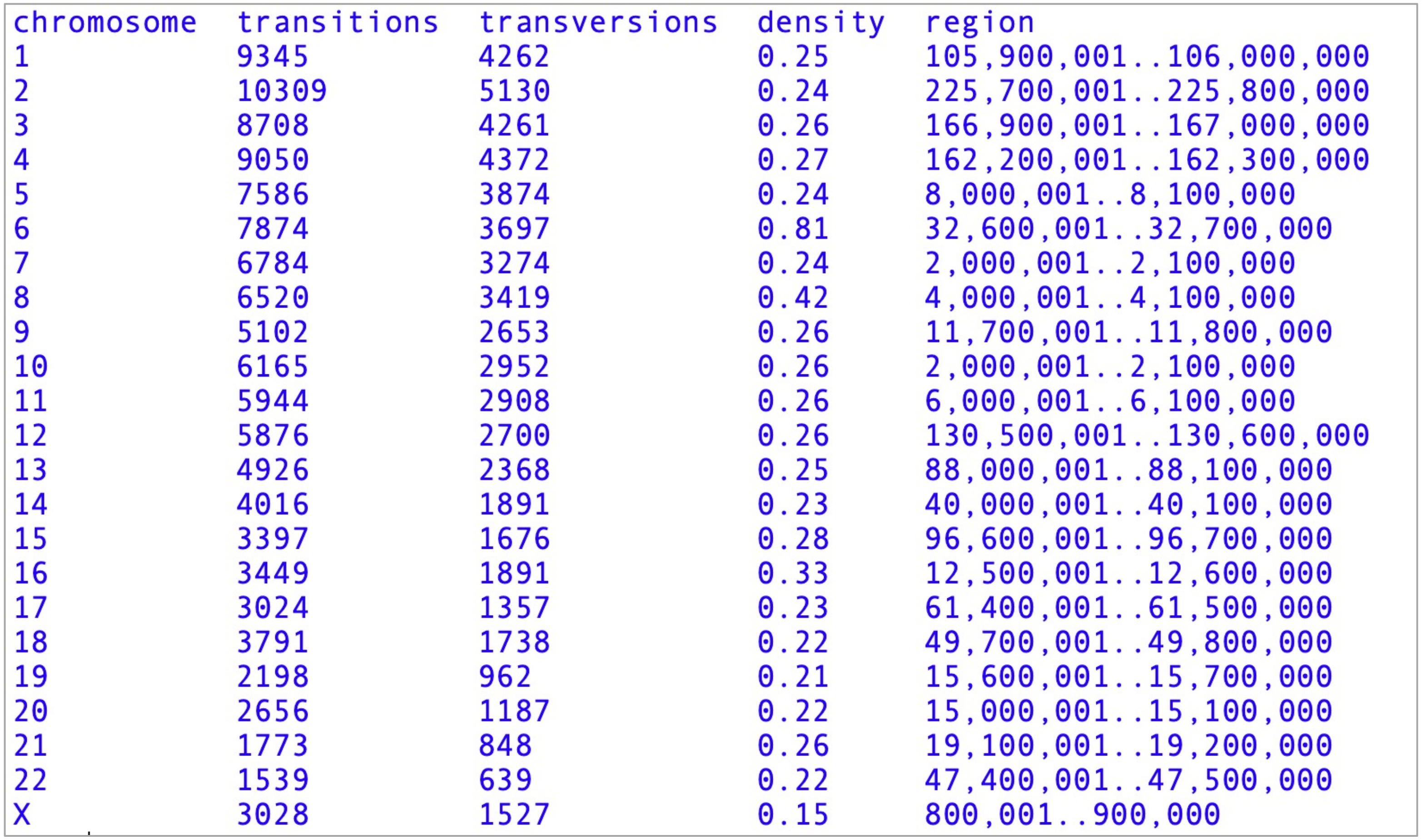
Inside Chromosome class

```
# (Inside Chromosome class ...)

# Returns the number of snps between l and m, divided by region size
def density_region(self, l, m):
    count = 0
    for location in self.locations_to_snps:
        if location >= l and location <= m:
            count += 1
    return 1000*count/float(m-l+1)


# Given a region size, looks at non-overlapping windows
# of that size and returns a list of three elements for
# the region with the highest density:
# [density of region, start of region, end of region]
def max_density(self, region_size):
    region_start = 1
    last_snp_position = max(self.locations_to_snps.keys())
    best_answer = [0.0, 1, region_size-1]

    while region_start < last_snp_position:
        region_end = region_start + region_size - 1
        region_density = self.density_region(region_start, region_end)
        if region_density > best_answer[0]:
            best_answer = [region_density, region_start, region_end]
        region_start = region_start + region_size

    return best_answer
```

The `main()` function

```
def main():

    # Create chrnames_to_chrs dictionary, parse the input file
    chrnames_to_chrs = dict()
    filename = "trio.sample.vcf"
    with open(filename, "r") as fh:
        for line in fh:
            # Skip header lines, which starts with #
            if not line.startswith("#"):
                fields = line.strip().split("\t")
                chrname = fields[0]
                pos = int(fields[1])
                snpid = fields[2]
                ref = fields[3]
                alt = fields[4]

                # Put the data to the dictionary
                if chrname not in chrnames_to_chrs:
                    chrnames_to_chrs[chrname] = Chromosome(chrname)
                chrnames_to_chrs[chrname].add_snp(chrname, pos, snpid, ref, alt)

    ## Print the results!
    region_size = 100000
    print("chromosome  transitions  transversions  density  region")
    for chrname in chrnames_to_chrs:
        chr_obj = chrnames_to_chrs[chrname]
        trs = chr_obj.count_transitions()
        trv = chr_obj.count_transversions()
        (density, region_start, region_end) = chr_obj.max_density(region_size)
        print(f"{chrname:12s}{trs:<13d}{trv:<15d}{density:<9.2f}" +
              f"{region_start:,}..{region_end:,}")
```

Program output



```
chromosome  transitions  transversions  density  region
1           9345         4262           0.25     105,900,001..106,000,000
2           10309        5130           0.24     225,700,001..225,800,000
3           8708         4261           0.26     166,900,001..167,000,000
4           9050         4372           0.27     162,200,001..162,300,000
5           7586         3874           0.24     8,000,001..8,100,000
6           7874         3697           0.81     32,600,001..32,700,000
7           6784         3274           0.24     2,000,001..2,100,000
8           6520         3419           0.42     4,000,001..4,100,000
9           5102         2653           0.26     11,700,001..11,800,000
10          6165         2952           0.26     2,000,001..2,100,000
11          5944         2908           0.26     6,000,001..6,100,000
12          5876         2700           0.26     130,500,001..130,600,000
13          4926         2368           0.25     88,000,001..88,100,000
14          4016         1891           0.23     40,000,001..40,100,000
15          3397         1676           0.28     96,600,001..96,700,000
16          3449         1891           0.33     12,500,001..12,600,000
17          3024         1357           0.23     61,400,001..61,500,000
18          3791         1738           0.22     49,700,001..49,800,000
19          2198         962            0.21     15,600,001..15,700,000
20          2656         1187           0.22     15,000,001..15,100,000
21          1773         848            0.26     19,100,001..19,200,000
22          1539         639            0.22     47,400,001..47,500,000
X           3028         1527           0.15     800,001..900,000
```

Readings

- Chapter 23, Part II, A Primer for Computational Biology
- Chapter 10 & 11, Starting out with Python