

Reusable python code for genomics

Dan Andrews

School of Computing, ANU

John Curtin School of Medical Research, ANU



Australian
National
University

Why do genetics and genomics matter?



Australian
National
University

- Essential for precision medicine
- Better diagnosis of genetic diseases
- Better diagnosis == targeted therapies
- Examples:
 - Treatment of cancer patients
 - Treatment of rare genetic disease



Why do geneticists need programming?



Australian
National
University



```
7     if path:
8         self.file = open(os.path.join(path,
9             self.file.seek(0)
10            self.fingerprints.update(fp)
11
12    @classmethod
13    def from_settings(cls, settings):
14        debug = settings.get("debug", False)
15        return cls(job_dirs=settings["job_dirs"],
16                  request_fingerprint=request_fingerprint,
17                  request_seen=request_seen,
18                  settings=settings,
19                  write_to_file=write_to_file,
20                  write_to_stdout=write_to_stdout)
21
22    def request_seen(self, request):
23        fp = self.request_fingerprint(request)
24        if fp in self.fingerprints:
25            return True
26        self.fingerprints.add(fp)
27        if self.file:
28            self.file.write(fp + os.linesep)
29
30    def request_fingerprint(self, request):
31        return self._request_fingerprint(request)
```

- The modern geneticist can't move without being able to process information
- The human genome project wouldn't have been able to succeed without huge amounts of code to decipher the sequence data

Why is object oriented programming relevant?



Australian
National
University

- Object-oriented programming is relevant to all software engineering. Especially as it grows in scale.
- Code structure, organization and reusability.



You know about modules



Australian
National
University

```
from counter import count_letters as cl

example_counts = cl('example')

for letter in example_counts:
    print('Letter: ' + letter + " Count: " + str(counts[letter]))
```

- **import, from and as keywords**
- **Code organization, reuse and sharing**

You should know about classes too...

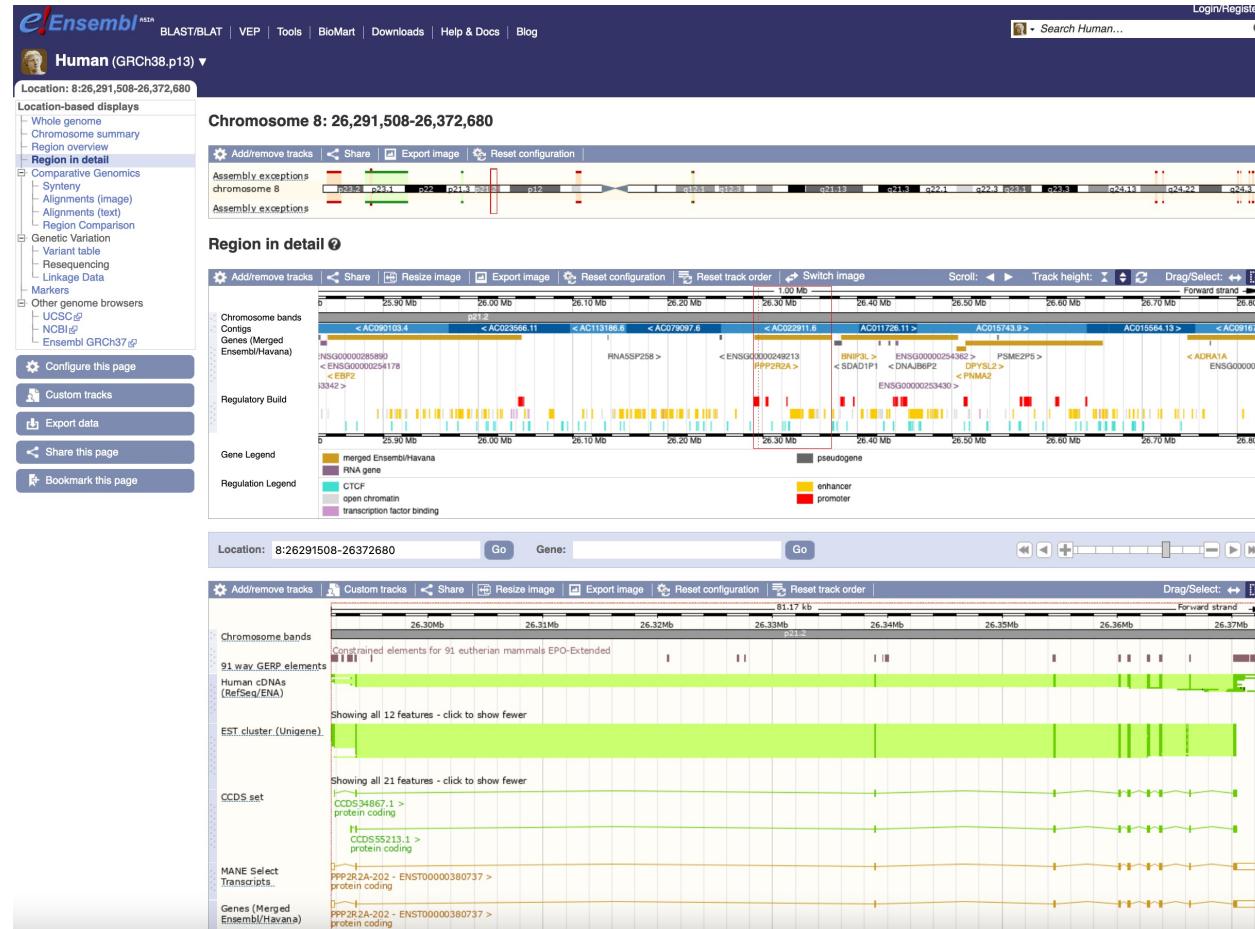
An anecdote



Australian
National
University

e!

ensembl.org



Genomes are cellular information systems

The Human Genome project sequenced DNA, the molecules that make up chromosomes in cells. The information derived from this project presented scientists with a valuable opportunity to not only uncover the secrets of DNA but also the manner in which genes are associated with disease. Scientists now are able to compare the genomes of people who have a certain condition with those who do not, in order to determine whether genetic variation plays a role in that condition. This information will help them to predict and possibly prevent disease in the future.

1. Cell
Each of the trillions of cells in the human body contains 46 chromosomes packed tightly into the region called the nucleus.

2. Chromosomes
Half of the chromosomes in the nucleus come from your mother, and half from your father. Each chromosome is a long, tightly coiled molecule called DNA, or deoxyribonucleic acid.

3. DNA
If unwound, the DNA from all the chromosomes in a single cell placed end to end would stretch more than six feet.

4. Genome
DNA is made up of chemical building blocks abbreviated A, C, T, and G. The entire length of a DNA strand consists of these four blocks in different combinations. Together, all the DNA in all the chromosomes – more than 3 billion letters – makes up the human genome. When scientists say they have "sequenced" the human genome, they mean that they have figured out the order of all those A's, C's, T's, and G's in sequence.

5. Genes: 30,000 DNA Segments
Much of the DNA in the genome is organized into units called genes. There may be as many as 30,000 genes in the genome; they are the instruction manual for making all the proteins in the body. These proteins are the physical "stuff" that makes up our hair, skin, heart, and blood, among other things. They also control chemical reactions, regulate blood sugar and heart rate, and control how food or medicine is metabolized in the body.

6. Misspellings in the Sequence
The way the genes are "spelled" makes all the difference - one letter out of place in a gene can cause disease. Now that we know the normal sequence of the human genome, researchers can compare the DNA sequence from people who have a disease or condition to those who don't. If there are differences in the spelling of certain genes between the two groups, it's possible that the condition may be caused by or related to that misspelling in that gene.

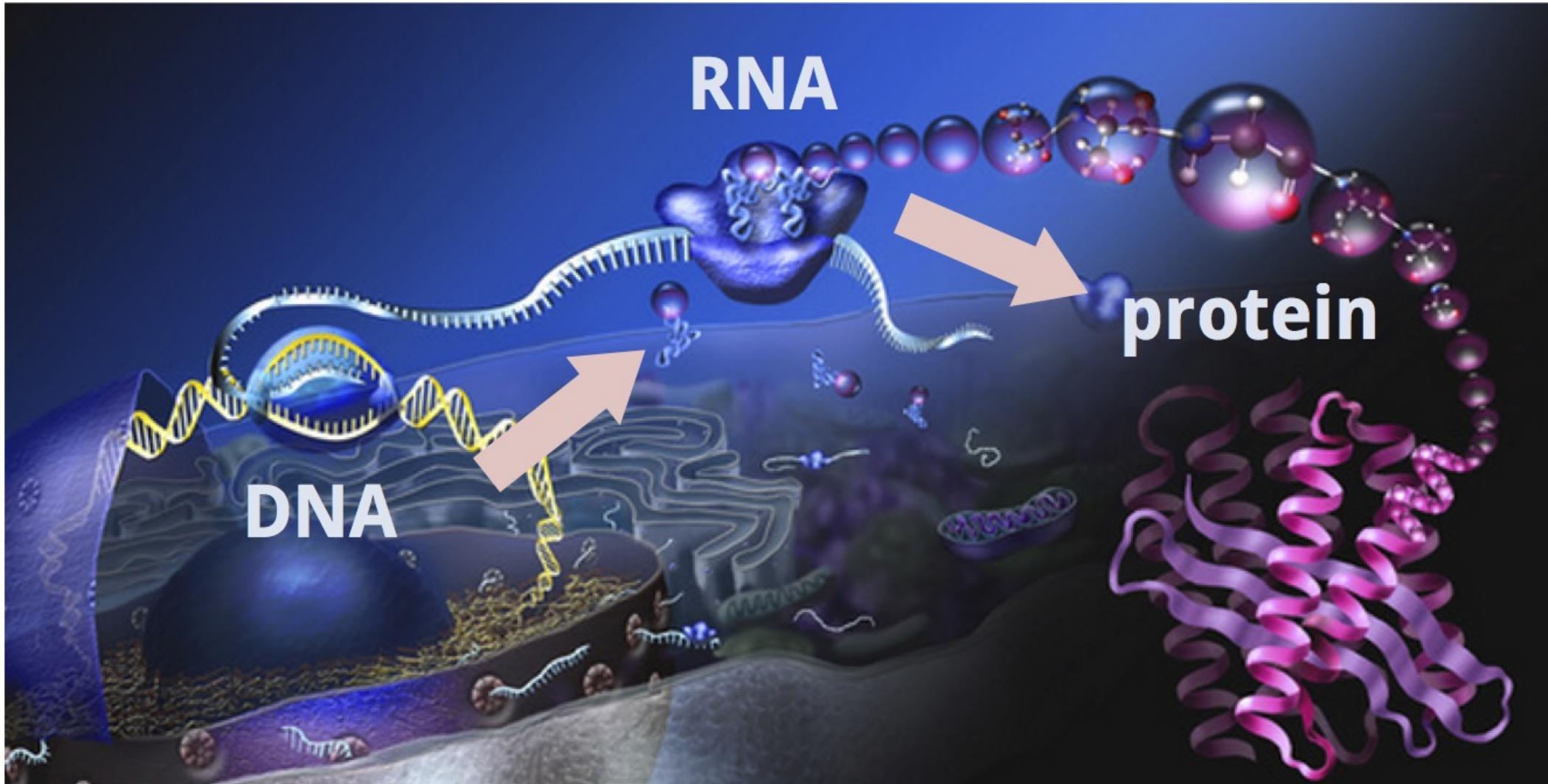
7. Genes and Disease
Scientists have identified about 6000 diseases, such as Huntington disease and cystic fibrosis, that are directly caused by misspellings or physical problems in single genes. But the genetic contribution to many common conditions – such as diabetes and heart disease – is part of a larger puzzle that could include diet, lifestyle, environment, and even other genes. For many of these common conditions, genetic misspellings probably make only a small contribution to disease relative to other factors, or work in concert with them to cause illness.



Australian
National
University

- Cells
- Nucleus
- Chromosomes
- DNA & Nucleotides
- Genes
- Mutations & Variation
- Disease

Genomes are cellular information systems



The Central Dogma of Molecular Biology

Genetic variation

nature

Vol 464 | April 2010 | doi:10.1038/nature08516

ARTICLES

Origins and functional impact of copy number variation in the human genome

Donald F. Conrad^{1*}, Dalila Pinto^{2*}, Richard Redon^{1,3}, Lars Feuk^{2,4}, Omer Gokcumen⁵, Yujun Zhang¹, Jan Aerts¹, T. Daniel Andrews¹, Chris Barnes¹, Peter Campbell¹, Tomas Fitzgerald¹, Min Hu¹, Chun Hwa Ihm⁵,

Vol 444 | 23 November 2006 | doi:10.1038/nature05329

nature

ARTICLES

Global variation in copy number in the human genome

Richard Redon¹, Shumei Ishikawa^{2,3}, Karen R. Fitch⁴, Lars Feuk^{5,6}, George H. Perry⁷, T. Daniel Andrews¹, Hei Fiegler¹, Michael H. Shapero⁴, Andrew R. Carson^{5,6}, Wenwei Chen⁴, Eun Kyung Cho⁷, Stephanie Dallaire⁷, Jennifer L. Freeman⁷, Juan R. González⁸, Mònica Gratacós⁸, Jing Huang⁹, Dimitrios Kalaitzopoulos¹, Daisuke Komura³, Jeffrey R. MacDonald⁵, Christian R. Marshall^{5,6}, Rui Mei⁴, Lyndal Montgomery¹, Kunihiro Nishimura², Kohji Okamura^{5,6}, Fan Shen⁴, Martin J. Somerville⁹, Joelle Tchinda⁷, Armand Valsesia¹, Cara Woodward¹, Fengtang Yang¹, Junjun Zhang⁹, Tatiana Zerjal¹, Jane Zhang⁴, Lluís Armengol⁸, Donald F. Conrad¹⁰, Xavier Estivill^{8,11}, Chris Tyler-Smith¹, Nigel P. Carter¹, Hiroyuki Aburatani^{2,12}, Charles Lee^{7,13}, Keith W. Jones⁴, Stephen W. Scherer^{5,6} & Matthew E. Hurles¹

Copy number variation (CNV) of DNA sequences is functionally significant but has yet to be fully ascertained. We have constructed a first-generation CNV map of the human genome through the study of 270 individuals from four populations with ancestry in Europe, Africa or Asia (the HapMap collection). DNA from these individuals was screened for CNV using two complementary technologies: single-nucleotide polymorphism (SNP) genotyping arrays, and clone-based comparative genomic hybridization. A total of 1,447 copy number variable regions (CNVRs), which can encompass overlapping or adjacent gains or losses, covering 360 megabases (12% of the genome) were identified in these populations. These CNVRs contained hundreds of genes, disease loci, functional elements and segmental duplications. Notably, the CNVRs encompassed more nucleotide content per genome than SNPs, underscoring the importance of CNV in genetic diversity and evolution. The data obtained delineate linkage disequilibrium patterns for many CNVs, and reveal marked variation in copy number among populations. We also demonstrate the utility of this resource for genetic disease studies.

Andy Wing Chun Pang²,
come Trust Case Control
r^{2,6} & Matthew E. Hurles¹

among human genomes, but are
; 42 million probes, to generate a
of which most (8,599) have been
n 450 individuals of European,
size classes. Retrotransposition
l the genome. Furthermore, by
ified 30 loci with CNVs that are
teness of our map and the
raits, the heritability void left by

Genetic variation staggers scientists

By Steve Connor

Scientists have discovered a dramatic variation in the genetic make-up of humans that could lead to a fundamental reappraisal of what causes incurable diseases and could provide a greater understanding of mankind.

The discovery has astonished scientists studying the human genome — the genetic recipe of mankind. Until now it was believed the variation between people was due largely to differences in the sequences of the individual “letters” of the genome.

It now appears much of the variation is explained instead by people having multiple copies of some key genes that make up the human genome. Until now it was assumed that the human genome, or “book of life”, was largely the same for everyone, save for a few spelling differences in some of the words. Instead, the findings suggest that the book contains entire sentences, paragraphs or even whole pages that are repeated any number of times.



Australian
National
University

The findings mean that instead of humanity being 99.9 per cent identical, as previously believed, we are at least 10 times more diverse than once thought — which could explain why some people are prone to serious diseases. The studies have found that instead of having just two copies of each gene — one from each parent — people can carry many copies, but just how many can vary between one person and the next. They suggest variations in the number of copies of genes is normal and healthy. But the scientists also believe many diseases may be triggered by an abnormal loss or gain in the copies of some key genes.

Another implication of the finding is that we have a greater divergence from our closest living relative, the chimpanzee, than previously assumed from earlier studies. Instead of being 99 per cent similar, we are more likely to be about 96 per cent similar.

The findings, published simultaneously in three leading science journals by scientists from 13 different research centres in Britain and America, have been described as ground-breaking.

A world authority on medical genetics at the Baylor College of Medicine in Houston, Texas, Professor James Lupski, said, “I believe this research will change for ever the field of human genetics.”

.....
Continued Page 2

Canberra Times, front page, Feb 2008

Precision Medicine

- All good, but we presently only solve 30-40% of cases
- The problem is that there is SO MUCH variation in a human genome



We are based at the John Curtin School of Medical Research, the Australian National University in Canberra. We work collaboratively with research groups and clinicians nationally and internationally.



What is the computation in genomics?



Australian
National
University

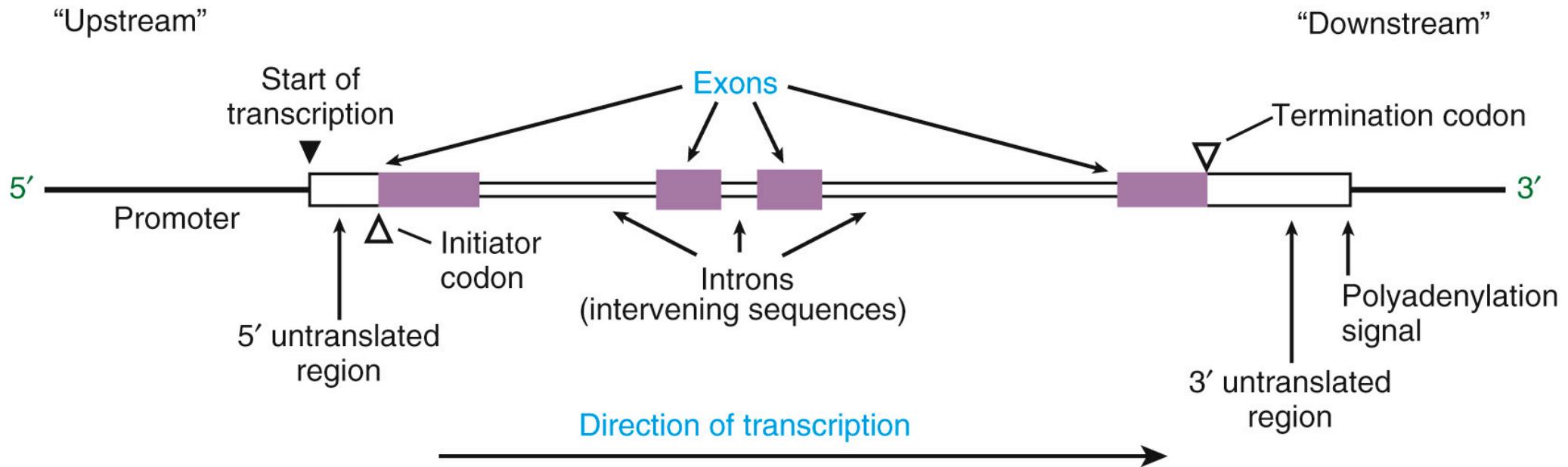
- Massive quantities of sequence data. Not just from one person, but very many people. 100k+
 - Writing computational pipelines to analyse data, even just for counting sequence variation and prioritising disease-causing mutations
- Sequence aligners and variant callers
- Prediction of functional effects of sequence variation
- Data integration and annotation, such that significance of sequence variation is better understood
- Common software tools in genomics (mostly open-source):
 - Genome Analysis Toolkit (GATK), SAMTools, BWA, VCFtools, SpliceAI, Variant Effect Predictor (VEP), BioMart, EnsEMBL, Bowtie, Bbmap, BLAST, and many more

3+ Gigabases of this



Australian
National
University

But we know a common pattern for genes



From Thompson & Thompson (2016) Genetics in Medicine, Ch. 3

Figure 3-4

A, General structure of a typical human gene. Individual labeled features are discussed in the text. B, Examples of three medically important human genes. Different mutations in the β -globin gene, with three exons, cause a variety of important disorders of hemoglobin ([Cases 42 and 44](#)). Mutations in the $BRCA1$ gene (24 exons) are responsible for many cases of inherited breast or breast and ovarian cancer ([Case 7](#)). Mutations in the β -myosin heavy chain ($MYH7$) gene (40 exons) lead to inherited hypertrophic cardiomyopathy.

And this allows predicted gene structures...



Australian
National
University

And then sequencing technology had a step change

- Can sequence 18000 whole genomes a year
- ~10X faster than current machines at 1/3 the cost
- Assuming 250Gb per 30x genome = 4.5Pb pa

Data production monsters





Don't laugh!

What is the data?

- Fastq data:

```
@HWI-EAS209_0006_FC706VJ:5:58:5894:21141#ATCACG/1
TTAATTGGTAAATAAATCTCCTAATAGCTTAGATNTTACCTNNNNNNNNNTAGTT
TCTTGAGATTGTTGGGGGAGACATTTGTGATTGCCTTGAT +HWI-
EAS209_0006_FC706VJ:5:58:5894:21141#ATCACG/1
efcfffffcfeffffcfffffd`feed] `]_Ba_ ^ __ [YBBBBBBBBBBRTT\]
] [] dddd`ddd^dddadd^BBBBBBBBBBBBBBBBBBBBBBBBBBB
```

- SAM/BAM data: text or binary, aligned to reference genome sequence
- VCF/BCF: variant calls between personal genome and reference genome

```
##FORMAT<ID=DP,Number=1>Type=Integer>Description="Read Depth">
##FORMAT<ID=HQ,Number=2>Type=Integer>Description="Haplotype Quality">
#CHROM POS ID REF ALT QUAL FILTER INFO FORMAT Sample1 Sample2 Sample3
2 4370 rs6057 G A 29 . NS=2;DP=13;AF=0.5;DB;H2 GT:GQ:DP:HQ 0|0:48:1:52,51
1|0:48:8:51,51 1/1:43:5:.,. 2 7330 . T A 3 q10 NS=5;DP=12;AF=0.017
GT:GQ:DP:HQ 0|0:46:3:58,50 0|1:3:5:65,3 0/0:41:3 2 110696 rs6055 A G,T 67
PASS NS=2;DP=10;AF=0.333,0.667;AA=T;DB GT:GQ:DP:HQ 1|2:21:6:23,27
2|1:2:0:18,2 2/2:35:4
```

It is no longer sensible to code stuff up alone!



Australian
National
University

- Current largest public datasets of genomes are 100k+ individuals
- Output of analysis tools is complicated
 - And your home-made VCF parser probably isn't as good as the community tools
- Use and contribute to open software libraries – which are most likely written in python.
- This isn't genomics-specific. Just about every scientific community with large datasets have open source libraries for handling this information.
- Doing otherwise is like being a machine learning researcher who still implements everything from scratch, rather than using SciKit-Learn, Keras or TensorFlow
- So you need to know about classes...

Classes

COMP1730/6730

Reading: Chapter 8, Sundnes, *Introduction to Scientific Programming with Python*



Australian
National
University

Also recommended: Chapter 10, Lubanovic, *Introducing Python*, 2nd Edition (2019)

Python is object-oriented



Australian
National
University

- You may hear repeatedly that everything in python is an object
 - The handling of lot of these objects is hidden by python syntax
- What is an object?
 - An object is a data structure that contains:
 - data (or values – sometimes called attributes)
 - code (functions – called methods in classes)
 - An object is defined by a **class**
- Why should you care?
 - In python, a lot of the time, you don't really need to care
 - Objects and the classes that define them can be a powerful way to organize code
 - The code libraries that make python particularly useful are implemented as classes
 - **instantiate** classes as objects when you use them
- Code that is large and/or complex is best implemented with classes

Classes (*Introducing Python Ch10*)



Australian
National
University

- You have seen modules and importing useful code from these
- At an introductory level, classes are just an extra formalism that makes things neater and more elegant.
- Object oriented coding is built on the class, but beyond the scope of this course
- But – knowing what a class is, and maybe how to write some simple ones for yourself, will make it easier to understand what external software libraries are all about.

Class vs module?



Australian
National
University

- When to use which?
 - Do you just need to import a function? Use modules
 - Do you need functions to operate in the context of some data? Use a class
- Instances: modules only allow a singleton. Classes can have multiple instances, that can all hold different attributes
- A class:
 - Can be instantiated with specific parameters
 - Is easily instantiated as many times as necessary
 - Contains methods that run on the particular instance of that class
 - Supports inheritance, polymorphism and all the object oriented stuff
- Modules become annoying if you:
 - Need to pass lots of data arguments to the function
 - Need to call many functions on the same parameter data

class definition syntax



Australian
National
University

- class keyword and a class name followed by a code block of a single line is the bare minimum (this does nothing, of course):

```
>>> class Cat():  
...     pass
```

- Then you **instantiate** an object of this class with:

```
>>> a_cat = Cat()  
>>> another_cat = Cat()
```

Lubanovic (2019) *Introducing Python*, Ch. 10

- These are distinct objects, from each instantiation. They have separate memory addresses

A class with one attribute - initialisation



Australian
National
University

- A class is often initialized at creation (instantiation). This is done by a special function named `__init__` :

```
>> class TeenyClass():
...     def __init__(self, name):
...         self.name = name
...
>>> teeny = TeenyClass('itsy')
>>> teeny.name
'itsy'
```

Lubanovic (2019) *Introducing Python*, Ch. 10

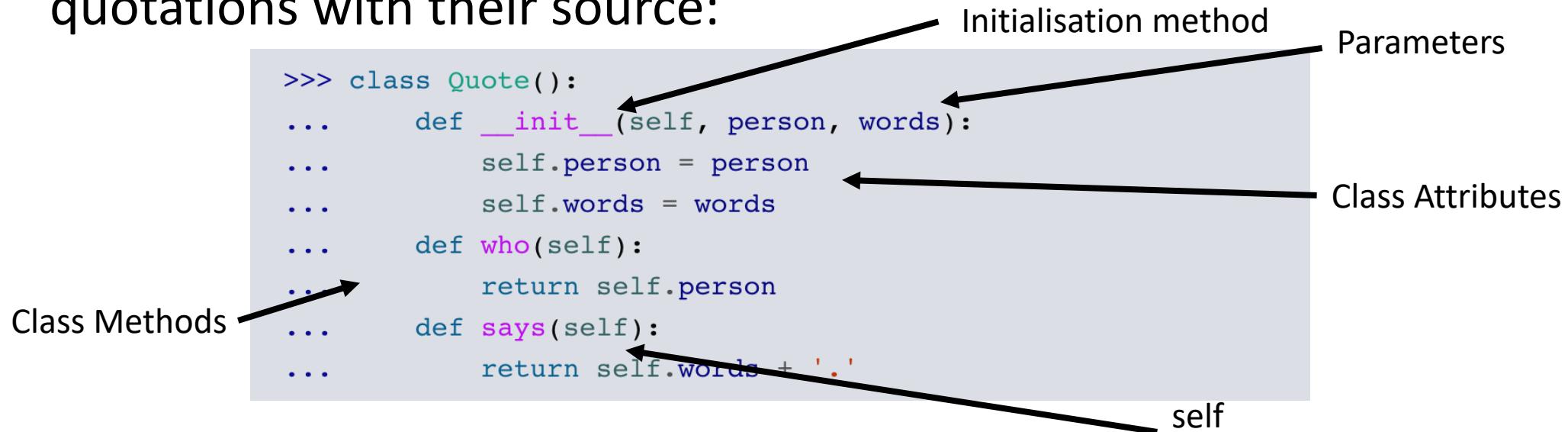
- The `__init__` function is optional, but may be defined to internally assign class attributes from parameter values (and many other things).

Anatomy of a class definition



Australian
National
University

- Say that your research interests required you to be able to associate quotations with their source:



- You could use instances of this class to access this information in your programs:

```
>>> hunter = Quote('Elmer Fudd', "I'm hunting wabbits")
>>> print(hunter.who(), 'says:', hunter.says())
Elmer Fudd says: I'm hunting wabbits.
```

self



Australian
National
University

- What is self?
 - It isn't strictly a python keyword
 - This is a concept broader than python
- Some experimenting:

```
>>> class WhatIsSelf():
...     def __init__(self, name):
...         self.name = name
...         self.self_id = id(self)

>>> wis = WhatIsSelf('A Name')
>>> type(wis)
<class '__main__.WhatIsSelf'>
>>> id(wis)
140614751849392
>>> wis.self_id
140614751849392
```

self



- When an object is referring to itself, the `self` variable name is commonly used. Don't use `self` as a variable name elsewhere in your code.
- The `self` variable is always the first (silent) argument in all function calls (including `__init__`)
 - but it is automatic and implicit
 - You don't need to ever specify
 - It is included in method parameters and will therefore be a local variable to your function/method

```
>>> class Quote():
...     def __init__(self, person, words):
...         self.person = person
...         self.words = words
...     def who(self):
...         return self.person
...     def says(self):
...         return self.words + '.'
```

The diagram shows three black arrows originating from the word "self" and pointing to three different instances of "self" within the Python code. The first arrow points to the first "self" in the `__init__` method parameter list. The second arrow points to the `self` variable assignment in the same method. The third arrow points to the `self` variable used as a parameter in the `who` method definition.

Attributes



- Attributes are the data or values that an object of a class holds
 - Can be parameters copied at initialisation
 - Can be default values
 - Can be derived values computed with class methods
- There is an open door to access attributes in python
 - All object attribute values can be accessed using the dot notation
 - Assumes programmers have discipline (or know what they are doing)

```
>>> class Quote():
...     def __init__(self, person, words):
...         self.person = person
...         self.words = words
...     def who(self):
...         return self.person
...     def says(self):
...         return self.words + '.'
```

Parameters

Class Attributes

The `__init__` method



Australian
National
University

- `__init__` is called when a class is instantiated
- Initialises class data and can perform checks
- Can call class methods at initialization

```
>>> class Quote():
...     def __init__(self, person, words):
...         self.person = person
...         self.words = words
...     def who(self):
...         return self.person
...     def says(self):
...         return self.words + '.'
```

Parameters

Class Attributes

Class methods



Australian
National
University

- Along with the data a class object contains, the benefit is also that class methods can be called to do tasks with this information

```
>>> class Quote():
...     def __init__(self, person, words):
...         self.person = person
...         self.words = words
...     def who(self):
...         return self.person
...     def says(self):
...         return self.words + '.'
```

Class Methods



An example class: PatientMutations



- The best reason to implement a class is to hide some complexity, to allow your client code to be simpler and easier to read and write
- This example is a class that imports data (personal genome information from a patient) and allows some filtering/retrieval of mutations
- **patient_mutations** class metadata: `patient_id`
- **Genetic variation data table:** `gene_name`, `coordinate`, `mutant`, `homozygous`, `essential_gene`, `damage_score`
- This data allows filtering of genetic variation data to find the disease-causing mutation
- With lots of patients with the same disease, we commonly look for common mutations in the same gene

Patient mutations data files:

Comma-separated values (CSV) file format:

gene_name, chromosome, coord, ref_nucl, var_nucl, homozygous?, essential_category, damage_score



Australian
National
University

mutations_193864.csv

```
TNFRSF4,1,1213738,G,A,True,2,0.74
PDE6B,3,46579986,C,T,True,2,0.94
TDGF1,4,660603,T,A,True,2,0.85
NDUFA13,19,19526194,T,C,True,2,1.00
PHEX,X,22247940,G,A,True,5,0.94
```

mutations_658192.csv

```
TNFRSF4,1,1213738,G,A,True,2,0.74
ACVR,12,157774144,A,G,True,3,0.67
HINT1,5,131165096,C,G,False,4,0.62
BCKDHB,6,80273147,A,G,False,7,0.56
SLC4A1,17,44253151,G,A,True,1,0.81
```

mutations_239872.csv

```
TNFRSF4,1,1213738,G,A,True,2,0.74
PDE6B,3,46579986,C,T,True,2,0.94
AMPD3,11,10500245,C,T,True,2,0.90
MOCOS,18,36195283,G,C,True,1,0.99
PHEX,X,22247940,G,A,True,5,0.94
```

mutations_283745.csv

```
TNFRSF4,1,1213738,G,A,True,2,0.74
ACVR,12,157774144,A,G,True,3,0.67
BCKDHB,6,80273147,A,G,False,7,0.56
SMARCA2,9,2060867,C,T,True,3,0.88
KRT2,12,52651601,T,G,False,8,0.23
```

mutations_947631.csv

```
TNFRSF4,1,1213738,G,A,True,2,0.74
TDGF1,4,660603,T,A,True,2,0.85
EGR2,10,62813491,C,A,True,2,0.98
MOCOS,18,36195283,G,C,True,1,0.99
NDUFA13,19,19526194,T,C,True,2,1.00
```

PatientMutations in Genomics.py



Australian
National
University

- This illustrates the features of a class all together:
 - The class definition
 - A Docstring
 - `__init__` method
 - Class attributes
 - Class methods

```
import csv

class PatientMutations():
    ''' An example class that imports personal genome mutation data from a
    file and allows retrieval/filtering of this by three criteria '''

    def __init__(self, patient_id, mutations_filename):
        self.patient_id = patient_id # metadata to link to patient
        self.input_file = mutations_filename
        self.mutations_data = [] # initialise mutation data as a list here
        self.read_mutations_file() # method call to read from input file

    def read_mutations_file(self):
        input_file = open(self.input_file, 'r')
        mutations_csv = csv.reader(input_file)

        for mutation in mutations_csv:
            mutation.append(self.patient_id)
            # populate internal mutations data from input file
            self.mutations_data.append(mutation)

        input_file.close()

    def candidate_disease_mutations(self,
                                    essential_cutoff=2,
                                    require_homozygous=True,
                                    damage_score_cutoff=0.7):
        disease_mutations = []

        for mutation in self.mutations_data:
            # copies of mutation information for readability below
            is_homozygous = bool(mutation[5])
            essential_score = int(mutation[6])
            damage_score = float(mutation[7])

            if (is_homozygous
                and essential_score <= essential_cutoff
                and damage_score >= damage_score_cutoff):
                disease_mutations.append(mutation)

        return disease_mutations
```

scan_mutations.py



Australian
National
University

```
from Genomics import PatientMutations

patient1 = PatientMutations(193862, 'mutations_193864.csv')
patient2 = PatientMutations(283745, 'mutations_283745.csv')
patient3 = PatientMutations(947631, 'mutations_947631.csv')
patient4 = PatientMutations(239872, 'mutations_239872.csv')
patient5 = PatientMutations(658192, 'mutations_658192.csv')

patients = [patient1, patient2, patient3, patient4, patient5]
mutation_tally = dict()

for patient in patients:
    candidate_mutations = patient.candidate_disease_mutations()

    for candidate_mutation in candidate_mutations:
        gene_name = candidate_mutation[0]
        if gene_name in mutation_tally:
            mutation_tally[gene_name] += 1
        else:
            mutation_tally[gene_name] = 1

for mutant_gene in mutation_tally:
    print(mutant_gene + ': ' + str(mutation_tally[mutant_gene]))
```

```
In [51]: runfile('/Users/dan/scan_mutations.py', wdir='/Users/dan')
TNFRSF4: 5
PDE6B: 2
TDGF1: 2
NDUFA13: 2
EGR2: 1
MOCOS: 2
AMPD3: 1
SLC4A1: 1
```

Classes written by other people



Australian
National
University

- This is really what libraries are (mainly) composed of. You have all used one already:
 - robot.py contains several classes
 - You imported the robot module:

```
import robot
```

- Then some magic detected if the robot was plugged in (or the simulation is used):

```
In [1]: robot.init()
```

- Initialising the robot (without a robot) instantiated the SimRobot class from robot.py. If you have the robot, the RPCRobot class is instantiated.
- With either of these objects, you could use class methods to:

- lift_gripper()
- move_right()
- etc

```
In [2]: robot.drive_right()
```

```
In [3]: robot.lift_up()
```

```
In [4]: robot.gripper_to_open()
```

```
In [5]: robot.lift_down()
```

A look at a Robot class:

- This is the RPCRobot class that can be found in robot.py from the labs
- Class RPCRobot
- Global attribute defaults
- `__init__` method
- Methods:
 - `lift_up`
 - `lift_down`
 - `drive_right`
 - Etc...

```
class RPCRobot:  
    '''Robot class interfacing with ev3 via RPYC.'''  
    DEFAULT_DRIVE_RIGHT = 575  
    DEFAULT_DRIVE_LEFT = 600  
    DEFAULT_LIFT_UP = 220  
    DEFAULT_LIFT_DOWN = 200  
  
    def __init__(self, ip_address = "192.168.0.1"):  
        import rpyc  
  
        self.rpcconn = rpyc.classic.connect(ip_address)  
        self.ev3 = self.rpcconn.modules.ev3dev.ev3  
        self.battery = self.ev3.PowerSupply()  
        self.drive = self.ev3.LargeMotor('outB')  
        self.lift = self.ev3.LargeMotor('outD')  
        self.gripper = self.ev3.MediumMotor('outC')  
        self.sensor = self.ev3.ColorSensor()  
        self.proxor = self.ev3.InfraredSensor()  
  
    def print_state(self):  
        print("drive at " + str(self.drive.position))  
        print("lift at " + str(self.lift.position))  
        print("gripper at " + str(self.gripper.position))  
        print("sensor read: " + str(self.sensor.value()))  
        print("proxor read: " + str(self.proxor.value()))  
        print("battery: " + str(self.battery.measured_volts) + "V, "  
             + str(self.battery.measured_amps) + "A")  
  
    # moving up doesn't require braking  
    def lift_up(self, distance=DEFAULT_LIFT_UP):  
        print("lift at " + str(self.lift.position)  
              + ", speed " + str(self.lift.speed))  
        self.lift.run_to_rel_pos(position_sp = -distance, duty_cycle_sp = -25)  
        time.sleep(0.5)  
        while abs(self.lift.speed) > 0:  
            print("lift at " + str(self.lift.position)  
                  + ", speed " + str(self.lift.speed))  
            time.sleep(0.25)  
        print("(end) lift at " + str(self.lift.position)  
              + ", speed " + str(self.lift.speed))  
  
    # moving down requires braking, and even then has to be commanded ~10 short  
    def lift_down(self, distance=DEFAULT_LIFT_DOWN):  
        print("lift at " + str(self.lift.position)  
              + ", speed " + str(self.lift.speed))  
        self.lift.run_to_rel_pos(position_sp = distance,  
                               duty_cycle_sp = 25,  
                               stop_command='brake')  
        time.sleep(0.5)  
        while abs(self.lift.speed) > 0:  
            print("lift at " + str(self.lift.position)  
                  + ", speed " + str(self.lift.speed))  
            time.sleep(0.25)  
        print("(end) lift at " + str(self.lift.position)  
              + ", speed " + str(self.lift.speed))  
  
    def drive_right(self, distance = DEFAULT_DRIVE_RIGHT):  
        print("drive at " + str(self.drive.position)  
              + ", speed " + str(self.drive.speed))  
        self.drive.run_to_rel_pos(position_sp = distance,  
                               duty_cycle_sp = 50,  
                               stop_command='hold')  
        time.sleep(0.5)
```



Just the beginning...



Australian
National
University

- Object-oriented coding is beyond the scope of this course, and could take up a whole course by itself
- Introducing classes is a starting point into object-oriented coding
- Further topics:
 - Magic methods
 - Inheritance
 - Polymorphism
 - Operator overloading
- You are encouraged to read more about them, if you are interested!