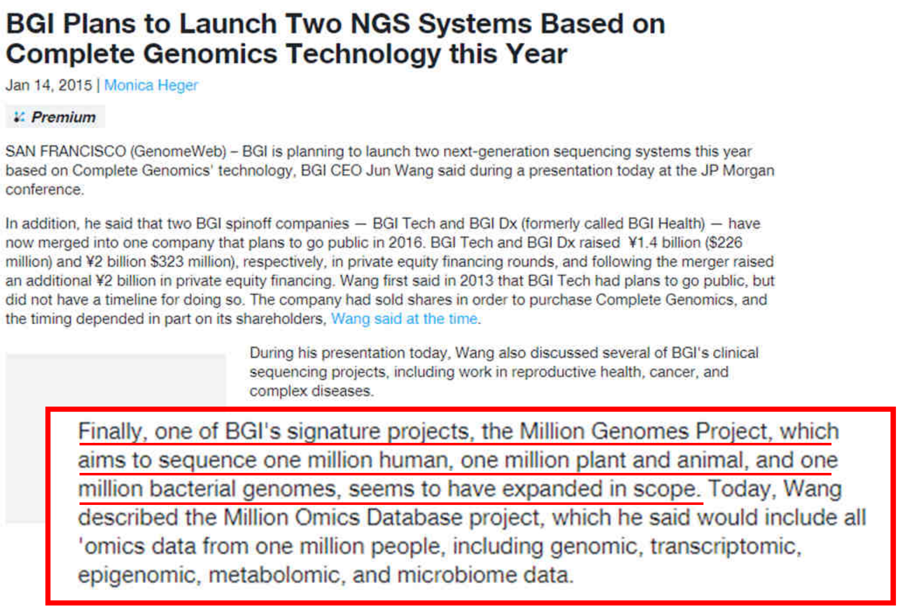
A Python programming primer for biochemists

(Named after *Monty Python's Flying Circus* & designed to be fun to use)

BCH339N Systems Biology/Bioinformatics
Edward Marcotte, Univ of Texas at Austin

Science news of the day (2015 edition):



BGI Plans to Launch Two NGS Systems Based on Complete Genomics Technology this Year

Jan 14, 2015 | Monica Heger

Premium

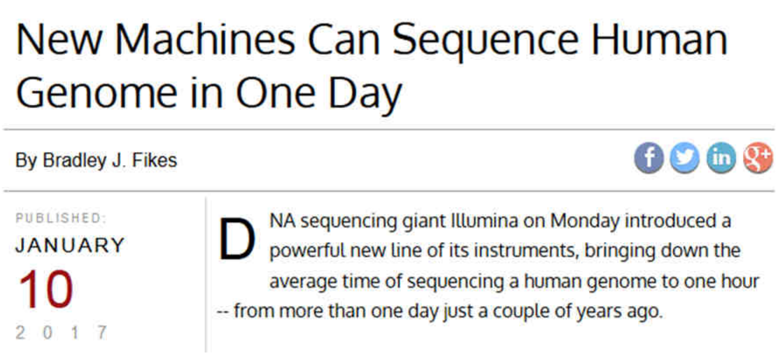
SAN FRANCISCO (GenomeWeb) – BGI is planning to launch two next-generation sequencing systems this year based on Complete Genomics' technology, BGI CEO Jun Wang said during a presentation today at the JP Morgan conference.

In addition, he said that two BGI spinoff companies — BGI Tech and BGI Dx (formerly called BGI Health) — have now merged into one company that plans to go public in 2016. BGI Tech and BGI Dx raised ¥1.4 billion ($226 million) and ¥2 billion $323 million), respectively, in private equity financing rounds, and following the merger raised an additional ¥2 billion in private equity financing. Wang first said in 2013 that BGI Tech had plans to go public, but did not have a timeline for doing so. The company had sold shares in order to purchase Complete Genomics, and the timing depended in part on its shareholders, Wang said at the time.

During his presentation today, Wang also discussed several of BGI's clinical sequencing projects, including work in reproductive health, cancer, and complex diseases.

Finally, one of BGI's signature projects, the Million Genomes Project, which aims to sequence one million human, one million plant and animal, and one million bacterial genomes, seems to have expanded in scope. Today, Wang described the Million Omics Database project, which he said would include all 'omics data from one million people, including genomic, transcriptomic, epigenomic, metabolomic, and microbiome data.

Science news of the day (2017 update):



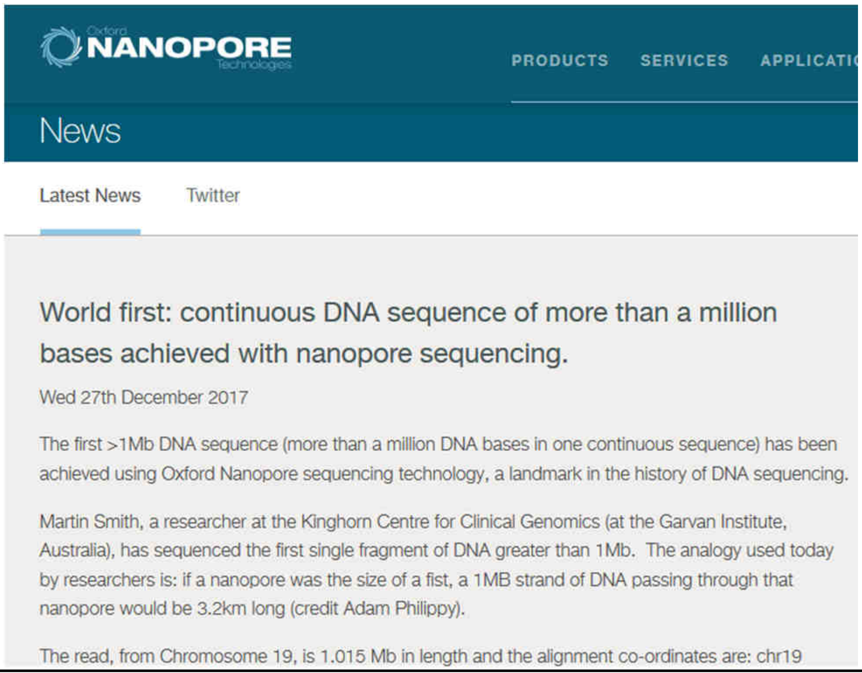
New Machines Can Sequence Human Genome in One Day

By Bradley J. Fikes

PUBLISHED: JANUARY 10 2017

DNA sequencing giant Illumina on Monday introduced a powerful new line of its instruments, bringing down the average time of sequencing a human genome to one hour -- from more than one day just a couple of years ago.

Science news of the day (2018 update):



World first: continuous DNA sequence of more than a million bases achieved with nanopore sequencing.

Wed 27th December 2017

The first >1Mb DNA sequence (more than a million DNA bases in one continuous sequence) has been achieved using Oxford Nanopore sequencing technology, a landmark in the history of DNA sequencing.

Martin Smith, a researcher at the Kinghorn Centre for Clinical Genomics (at the Garvan Institute, Australia), has sequenced the first single fragment of DNA greater than 1Mb. The analogy used today by researchers is: if a nanopore was the size of a fist, a 1MB strand of DNA passing through that nanopore would be 3.2km long (credit Adam Philippy).

The read, from Chromosome 19, is 1.015 Mb in length and the alignment co-ordinates are: chr19

In bioinformatics, you often want to do completely new analyses. Having the ability to program a computer opens up all sorts of research opportunities. Plus, it's fun.

Most bioinformatics researchers use a scripting language, such as Python, Perl, or Ruby.

These languages are not the fastest, not the slowest, nor best, nor worst languages, but they're easy to learn and write, and for many reasons, are well-suited to bioinformatics.

We'll spend the next 2 lectures giving an introduction to Python. This will give you a sense for the language and help us introduce the basics of algorithms

Python documentation: http://www.python.org/doc/
& tips: http://www.tutorialspoint.com/python

Good introductory Python books:
Learning Python, Mark Lutz & David Ascher, O'Reilly Media

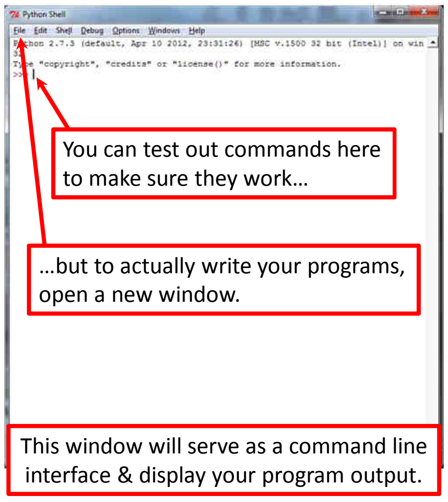
Bioinformatics Programming Using Python: Practical Programming for Biological Data, Mitchell L. Model, O'Reilly Media

Good intro videos on Python:
CodeAcademy: http://www.codecademy.com/tracks/python
& the Khan Academy:
https://www.khanacademy.org/science/computer-science

A bit more advanced: *Programming Python*
Mark Lutz, O'Reilly Media

By now, you should have installed Python on your computer, following the instructions in Rosalind Homework problem #1.

Launch IDLE:

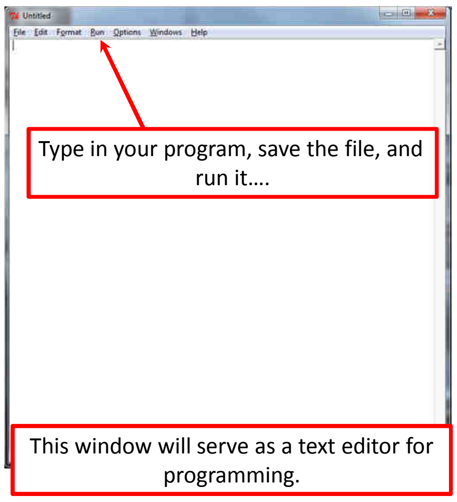


```
Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> |
```

You can test out commands here to make sure they work…

…but to actually write your programs, open a new window.

This window will serve as a command line interface & display your program output.



Type in your program, save the file, and run it….

This window will serve as a text editor for programming.

Let's start with some simple programs in Python:

A very simple example is:

```
print("Hello, future bioinformatician!")     # print out the greeting
```

Let's call it hello.py
Save & run the program. The output looks like this:

```
Hello, future bioinformatician!
```

A slightly more sophisticated version:

```
name = raw_input("What is your name? ")  # asks a question and saves the answer
                                          # in the variable "name"
print("Hello, future bioinformatician " + name + "!")       # print out the greeting
```

When you run it this time, the output looks like:

```
What is your name?
```

If you type in your name, followed by the enter key, the program will print:

```
Hello, future bioinformatician Alice!
```

GENERAL CONCEPTS

Names, numbers, words, etc. are stored as ***variables***.

Variables in Python can be named essentially anything except words Python uses as command.

For example:

```
BobsSocialSecurityNumber = 456249685
mole = 6.022e-23
password = "7 infinite fields of blue"
```

Note that strings of letters and/or numbers are in quotes, unlike numerical values.

LISTS

Groups of variables can be stored as lists.
A list is a <u>numbered</u> series of values,
like a <u>vector</u>, an <u>array</u>, or a <u>matrix</u>.

Lists are variables, so you can name them just as you would name any other variable.

Individual elements of the list can be referred to using [] notation:

The list nucleotides might contain the elements
nucleotides[0] = "A"
nucleotides[1] = "C"
nucleotides[2] = "G"
nucleotides[3] = "T"

(Notice the numbering starts from zero. This is standard in Python.)

DICTIONARIES

A VERY useful variation on lists is called a ***dictionary*** or *dict*
(sometimes also called a *hash*).

→ Groups of values indexed not with numbers (although they could be) but with other values.

Individual hash elements are accessed like array elements:

For example, we could store the genetic code in a hash named *codons*, which might contain 64 entries, one for each codon, e.g.

codons["ATG"] = "Methionine"
codons["TAG"] = "Stop codon"
etc…

Now, for some control over what happens in programs.

There are two very important ways to control the logical flow of your programs:

if statements

and

for loops

There are some other ways too, but this will get you going for now.

if *statements*

```
if dnaTriplet == "ATG":
        # Start translating here.  We're not going to write this part
        # since we're really just learning about IF statements
else:
        # Read another codon
```

Python cares about the white space (tabs & spaces) you use! This is how it knows where the conditional actions that follow begin and end. **These conditional steps must *always* be indented by the same number of spaces (e.g., 4).**

I recommend using a tab (rather than spaces) so you're always consistent.

Note: in the sense of performing a comparison, not as in setting a value.

==	equals
!=	is not equal to
<	is less than
>	is greater than
<=	is less than or equal to
>=	is greater than or equal to

Can nest these using parentheses and Boolean operations, such as *and, not*, or *or*, e.g.:

```
if dnaTriplet == "TAA" or dnaTriplet == "TAG" or dnaTriplet == "TGA":
    print("Reached stop codon")
```

for *loops*

Often, we'd like to perform the same command repeatedly or with slight variations.

For example, to calculate the mean value of the number in an array, we might try:

```
Take each value in the array in turn.
Add each value to a running sum.
Divide the total by the number of values.
```

In Python, you could write this as:

```
grades = [93, 95, 87, 63, 75]   # create a list of grades
sum = 0.0                       # variable to store the sum

for grade in grades:            # i
        sum = sum + grade      # i
                                # e

mean = sum / 5             # now calculate the average grade

print ("The average grade is "),mean   # print the results
```

Python cares whether numbers are **integers** or **floating point** (also **long integers** and **complex numbers**).
Tell Python you want floating point by defining your variables accordingly (e.g., X = 1.0 versus X = 1)

In general, Python will perform most mathematical operations, e.g.

multiplication	**(A * B)**
division	**(A / B)**
exponentiation	**(A ** B)**
etc.	

There are lots of advanced mathematical capabilities you can explore later on.

READING FILES

You can use a *for* loop to read text files line by line:

Stands for “read”

```
count = 0                                # Declare a variable to count lines
file = open("mygenomefile", "r")         # Open a file for reading (r)
for raw_line in file:                    # Loop through each line in the file
    line = raw_line.rstrip("\r\n")               wline
    words = line.split(" ")              # split the line into a list of words

    # Print the appropriate word:
    print "The first word of line {0} of the file is {1}".format(count, words[0])
    count += 1                           # shorthand for count = count + 1

file.close(                              # Last, close the file.
print "Read in {0} lines\n".format(count)
```

\r = carriage return
\n = newline

Increment counter by 1

Placeholders (e.g., {0}) in the print statement indicate variables listed at the end of the line after the format command

WRITING FILES

Same as reading files, but use "w" for ‘write’:

```
file = open("test_file", "w")
file.write("Hello!\n")
file.write("Goodbye!\n")
file.close()                    # close the file as you did before
```

Unless you specify otherwise, you can find the new text file you created (test_file) in the default Python directory on your computer.

PUTTING IT ALL TOGETHER

```
seq_filename = "Ecoli_genome.txt"
total_length = 0
nucleotide = {}                         # create an empty dictionary

seq_file = open(seq_filename, "r")
for raw_line in seq_file:
    line = raw_line.rstrip("\r\n")
    length = len(line)                  # Python function to calculate the length of a string
    for nuc in line:
        if nucleotide.has_key(nuc):
            nucleotide[nuc] += 1
        else:
            nucleotide[nuc] = 1
    total_length += length

seq_file.close()

for n in nucleotide.keys():
    fraction = 100.0 * nucleotide[n] / total_length
    print "The nucleotide {0} occurs {1} times, or {2} %".format(n, nucleotide[n], fraction)
```

Let’s choose the input DNA sequence in the file to be the genome of *E. coli*, available from the **Entrez genomes** web site or the class web site.

The format of the file is ~77,000 lines of A’s, C’s, G’s and T’s:

```
AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTC
TGATAGCAGCTTCTGAACTGGTTACCTGCCGTGAGTAAATTAAAATTTTATTGACTTAGG
TCACTAAATACTTTAACCAATATAGGCATAGCGCACAGACAGATAAAAATTACAGAGTAC
ACAACATCCATGAAACGCATTAGCACCACCATTACCACCACCATCACCATTACCACAGGT
```

etc…

Running the program produces the output:

```
The nucleotide A occurs 1142136 times, or 24.6191332553 %
The nucleotide C occurs 1179433 times, or 25.423082884 %
The nucleotide T occurs 1140877 times, or 24.5919950785 %
The nucleotide G occurs 1176775 times, or 25.3657887822 %
```

So, now we know that the four nucleotides are present in roughly equal numbers in the *E. coli* genome.