**Week 2: Introduction to programming in python – 4th May**

Interpreted – means you can write in a human friendly way

Have to index/format your code properly – visually understand your code better

In terminal, open python environment and load Spyder (spyder is like rstudio)

>>> #python environment – now wont

$ #bash environment

Terminal text editors v IDE – integrated development environments

Left is file tree (like finder)

Debugging section – red bits are break points – can see variable at each point in time

Notebook – bit of code, some plot etc

Go to definition – can use cursor to look at the definition

Pep8 #like the rule book – gives you problems and troubleshooting

Jupyter ntebooks #lab book for quite data analysis – like Rmarkdown – good as they send it as a html

We will proably use JupyterLab in cgat

Jupyter supports all 3 langauges – can type R in same notebook

**Variables**

Letter/underscore numbers only – don’t use dots or –

Boolean # wants capital at start only – cn’t use the <- like in R

“string” ‘string’ #same thing

Can include new lines within a string and python recognises it

Operators are like +/- = etc

==#means exactly/equal

!= #opposite to what follows, using !

Condiitonals #if output is this, do that, if output is d, do this

If else and elif

elif #eek

for/loop/while

break #breaking clause - stops an infinite look

continue #skips part of a loop, ie as long as this is true then skip to this

Functions

Naming a block of statements – function -basically a shortcut

Def #definition

Is liking setting up alias/shortcuts

In python we use rounds brackets

Can use functions across scripts etc

Parameters #giving values to a function, ie give them values within some code – like algebra

Return #returns a value from a function

Returning is like translating

Print wont be able to take a variable and send it to a screen

On slide 17#

Look at example of slide 19 #

[] #means empty list – python starts from 0 rather than 1 – like indexing

Tuple #cannot change one it’s set

= #defines a variable

Len() #lengths of a list

Append #appends to end of a list

Sort a list alphabetically –

Lists of vairables – like symbolic links in that if you change the original

Can copy the list – or use tuples to confirm this

Dictionaries – like address book, variable could be email/phone no

Define list with [] tuple with () and dictionary with {}

Access variables in a dic by using square brackets containing name of variable you want

So:

Dictionaryname[‘variable’]

Get #will get you variable

Del #delete a variable in a dictionary

Deepcopy #copies the nested lists too – make sure you duplicate whole thing rather than 2D

Slide 30

When you open a file you have to say whether you want to read or write from the file

Use of “with” #opens file as f for reading – and you don’t need to close the file

Useful if code crashes and this is best practice

Manipulating stirngs – read a string from a file and then how to

Starts with

Usng if (looks for if true then do this…)

If you put a -2 then it searches the stirng from the 2nd to end

PEP8 -style guide

One statement per line, indentations are eforced (4 spaces) tab will do this for you in spyder

Underscores separate words

Two spaces after end of function

Modules – it’s like loading a library in R -should do at the start

Code organisation

Modules should be imported and not run on command line etc

Getcwd #pwd in python

Package is folder with code, documentation etc

Extra sources at the end o the slides

Exercies – use the console so far

Can hover over functions to look at their definitions

Type() #looks at tge type of a number

Help(range) #this you can see in the help section

Using a for loop to print each of the values between 1 to 9

for i in range(1,10,1):

print(i)

so I means for each value, print this

for i in range(1,10,1):

print(i\*\*2)

#this prints the square

To print the natural log of 10, had to do

import math

math.log(10,2) #prints the log of 10 in base 2

math.log(10,10)

math.log1p(1)

math.sqrt(1234)

round(0.05)

math.ceil(0.05)

math.floor(0.05)

#prints current date and time

datetime.datetime.now()

Logging – alias – shortcuts etc

Import logging as L

L.info(“reading data” ) #prints a log error – it’s a way of using the errors to keep a log – gives a way to log any deroors and gives you some more information, gives INFO: #Log

Can extract all the info with just the log

Debug messages are sent for every iteration

Logging is a way of debugging

How to debug your code, can use a debugger

Spyder – debug function, pause your code at a specifc line and inspect code by line

Using a scientific approach to make a bug reproducible – experiment with code, - too much data? Computer dependent? Memory lowe? Multi thread?

Bugs in other peoples code: Github

Document the bug, so what the conditions of when your bug arose so the developer can try and fix it by replicating

Notes in OBDScourse\_Day1.py

May 5th Week 2 – Biological algorithms

Good idea to write pseudocode explaining exactly what you want to do – ie. Plain English explaining what to do. Or could write a flow chart for a visual of what you want to do

In github, click on profile, go to ssh and gpg keys, new ssh key.

In local terminal:

cd .ssh

less rd\_psa.pub

copy rd\_psa.pub

paste into ssh key

Name ssh key with local computer name

git clone git@github.com:OBDS-Training/OBDS\_Training\_Apr\_2020.git

I made a lot of my notes in the File Week2\_py\_day2.py

Day 3

Argparse – look at the pseudocode

Call arp and ask for argument parser -

Day 4 – Object oriented programming

Use of pysam – manage genomic formats

Pandas – manipulate data frames

Objects v classes

Class – can describe an object but also the type of the object, so info about the data too

Hierarchical relationships of classes: different attributes

Encapsulation – how to access objects private data using methods

Abstraction – internal object functions, higher functions that have internal functions of this

Polymorphism – subclasses of common methods etc, so like getsize for both a triangle and a square

function = method if it's part of a class

Computation genomics tasks

Pysam – python version of sam tools, outsource the work to a c library

Classes that represent each file type

Fastq file: each read has 4 line: read identifier, sequence, +line and quality score/ASCII characters

Sanger/other seystms use different ASCII characters etc

Pysam converts the number to a quality score appropriate for you (slide 4)

Slide 5 has the code: saying with fastq file (as fh), then print name, sequence, comment and quality (just to read it – don’t need to split line etc as pysam does this for you)

Or can have an file in and file out

Name, sequence etc are public attributes of that class of object

SAM file format – all fields, all attributes – pysam can break down these fields individually

Read name: ID of read pair

Flag: can indicate whether both pairs mapped, gives you info on the reads

Position – ref sequence name (1 means chr 1) then position on chr

Mapping quality (quality of entire mapping – how confirdent is program on mapping quality)

Cigar

Mate info = 14 10

Read sequence

Quality score

Metadata

Slide 8

Iter : iterator, in that sam files, fetch me these alignments (between a to b in a sequence).

Iter is really just saying: just read the region between here and here, then a for loop for each bit of this region

Info on iteration:

<https://stackabuse.com/introduction-to-python-iterators/>

If you say the file is “-“ then could pipe a file from command line into the script

Fetch() #method – chrosome

Name of the methods –

Align segment – paired or unpaired #look at public attributes

Tags – optional outputs from program

Get\_overlap

Pile up - returns a base for each thing in an iterator

Specific type of object – attribute n – how many reads are attributed from that base

Slide 12:

Define header at top: sequence dictionary

Write a binary bam file: header is header (write to this file)

A is a pysam aligned segment, set the attributes – will write it

a.cigar #how do you write cigar string –

Samtools commands – can call samtools terminal tools using python

Argument is “-o” then give it output nd input

Genomic intervals

Pysam works with tabix files

<http://www.htslib.org/doc/tabix.html>

BED file format

11 and 12: How big the block is and 12 where the block starts

GFF files – how many read map to each gene, ie rnaseq

Browser lines are optional instructions

1. Name of chromosome
2. Source – program name
3. Feature – will tell you what it’s encoding (have to fill this field but can be anything)
4. Start based (inclusive
5. Stop base – have to include the base that it ends (not like python counting)
6. Score
7. Stand (can put . if you don’t care)
8. If first nucleotide of your feature is in reading frame, then 0, but if 2nd pos then 1 if later, 3
9. Groups-

GTF – same as GFF but extra rules (first 8 is same – but also has more features) 9th field, 2 of attributes have to be geneid and transcript id

Have to tabix file format

Tabix files

File has to be compressed and indexed, then pysam can read it – wrap it as tabix, then pysam can fetch a d run stuff

Asking pysam to return a bed object – but can read back how you want

<https://www.freecodecamp.org/news/object-oriented-programming-concepts-21bb035f7260/>

Exercises

Write a script to convert a bam fike to bed using he pysam tool

Use command line to run script (argparse wrapper)

Use stin and stout (is this bash? Submission?)

samtools view -S -b ERR1755082.test.sam > ERR1755082.test.bam #converts to bam

samtools sort ERR1755082.test.bam -o ERR1755082.test.sorted.bam #sorts bam

samtools index ERR1755082.test.sorted.bam #indexes bam