CHICAS
centre for health informatics, computing, and statistics

Python Programming Introduction

Barry Rowlingson

Lancaster University
Medical School

Why Python?

CHICAS centre for health informatics, computing, and statistics

- General Purpose Programming Language
- Free and Open Source
- Large Module Library
- Used Widely in Web Services
- Increasingly Used for Statistics and Data Science
  - Numpy, Scipy, TensorFlow, PyMC3...

Why Python? Well, its another of the great general purpose programming languages out there (see also Ruby, C, C++, Fortran, Go, Rust, JS, Java…) and its free and open source so we can use it with no restriction or payment. I has a large library of modules supplied with the language, and a vast community of people writing additional modules for it. Its used widely in web servers and also for statistics and data science via some of the numerical modules such as numpy, TensorFlow, PyMC3 etc!

In The Beginning...

CHICAS centre for health informatics, computing, and statistics

...was the command line.

How did Python come about? Well in the beginning was the command line. I started my research career working at one of these, a dumb text terminal. The computer was in another room, or another building, and we sent it a line of text at a time as a command. You typed the command, it went to the computer, the computer did its thing and the output text came back.

Repeating Yourself...

CHICAS centre for health informatics, computing, and statistics

```
> dosomething file1.txt
[processing file1.txt]
> dosomething file2.txt
[processing file2.txt]
> dosomething file3.txt
[processing file3.txt]
```

And the commands were dumb also. If you wanted to do something to a bunch of files, you had to type it all in again. From the start. And computer nerds, being lazy, and realising that computers were the best thing in the world for repeating stuff without complaining, weren't having that.

Repeating Yourself...

CHICAS centre for health informatics, computing, and statistics

```
> for n in 1 2 3 ; do
   dosomething file${n}.txt
  done
[processing file1.txt]
[processing file2.txt]
[processing file3.txt]
```

So they wrote languages for their commands. So instead of typing in a command 100 times, they could write a loop from 1 to 100 and change the command line according to the loop value.

Shell

CHICAS centre for health informatics, computing, and statistics

```
gcd() {
        # Calculate $1 % $2 until $2 becomes zero.
        until test 0 -eq "$2"; do
                # Parallel assignment: set -- 1 2
                set -- "$2" "`expr "$1" % "$2"`"
        done

        # Echo absolute value of $1.
        test 0 -gt "$1" && set -- "`expr 0 - "$1"`"
        echo "$1"
}

lcm() {
        set -- "$1" "$2" "`gcd "$1" "$2"`"
        set -- "`expr "$1" \* "$2" / "$3"`"
        test 0 -gt "$1" && set -- "`expr 0 - "$1"`"
        echo "$1"
}

lcm 30 -42
# => 210
```

This led to the family of Unix “shell” command languages. Modern ones can do things like this, this being functions for computing the greatest common divisor and the least common multiple of two integers. Shell languages have a lot of quirks and require a lot of knowledge and skill to build reliable programs that don’t break easily, partly because they are built on much simpler command systems and complexity is hard.

perl

CHICAS centre for health informatics, computing, and statistics

- Born: 1988
- Strengths:
  - Interpreted
  - Text stream processing
  - CPAN (1993)
  - Nothing else like it
- Weaknesses
  - Confusing Syntax
  - Nothing else like it

And then Larry Wall released The “perl” language as an interpreted language intended initially for taking input streams of text, processing it, and producing an output stream. This coincided with the rise of the WWW, and so lots of code for web servers was needed that took web requests, processed them, and produced web pages back to the users, and so a lot of that was done with perl. There was also CPAN, the Comprehensive Perl Archive Network, which grew the capabilities of the language through community contributions. There was at the time, nothing else like Perl. But with its sometimes confusing syntax, being unique was also a weakness.

lcm in perl

CHICAS centre for health informatics, computing, and statistics

```
sub lcm {
use integer;
my($x, $y)=@_;
my($f,$s)=@_;
while($f!=$s){
($f,$s,$x,$y)=($s,$f,$y,$x)if$f>$s;
$f= $s / $x * $x;
$f += $x if $f < $s;
}
$f
}
```

Lots of punctuation everywhere

Here's a chunk of perl that does the same as one of those shell code functions, namely compute the greatest common divisor. There's a lot of punctuation here (every line has to end with a semi-colon, except where it doesn't), and lots of quirky things like @_ and +=, and expressions with "if" at the end of them, which seems odd to anyone used to R.

Obfuscated perl

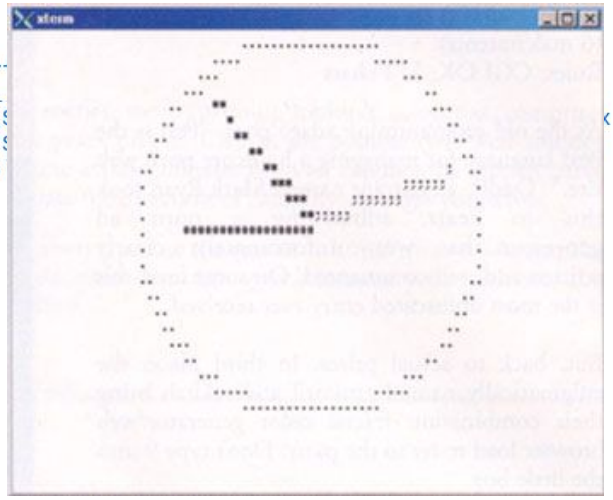
CHICAS centre for health informatics, computing, and statistics

```
#!/usr/bin/perl
$;="@{'`|;{'^'!.|-'}";$.++;$.++;$.++;$_="(.)?";/((?{$_.=$_}).)+$/;@_='~!@#$%^&*(
)_+`-=[]\\{}|;\':",./<>? '=~/$_/;@_ _=$;=~/$_/;$_="(.)*?";/((?{$_.=$_}).)+$/;$Z-=
$Z;"$.$."-$Z;/((?{$_ _[$z]&&!("${_[$x]}"^"${_[$y]}"^"${_ _[$z]}"^"$Z")&&($a.=$_[$x
],$b.=$_[$y],$z++);$x++;$y+=!($x%="$.$.");$y%="$.$.";}).)+/;$_="^"^"^";$_ _=".>.\
'$_ _ _$b')".".('!\@/\"'^'}.')".']}`';

print;
```

Perl's syntax meant you could write programs that look like this, and this was a winner from one of the contests to write the most obfuscated - ie where the true function of the code was hidden - perl programs. What does it do?

Obfuscated perl

```
#!/usr/bin/perl
$;="@{'`|;{'^'!.|-'}";$.++;$.++;$.+-
)_+`-=[]\\{}|;\':",./<>? '=~/$_/;@_
$Z;"$.$."-$Z;/((?{$_ _[$z]&&!("${_[$
],$b.=$_[$y],$z++);$x++;$y+=!($x%="$
'$_ _ _$b')".".('!\@/\"'^'}.')".']}

print;
```



It actually produces an animated clock, using characters for the hands. Genius, and insane.

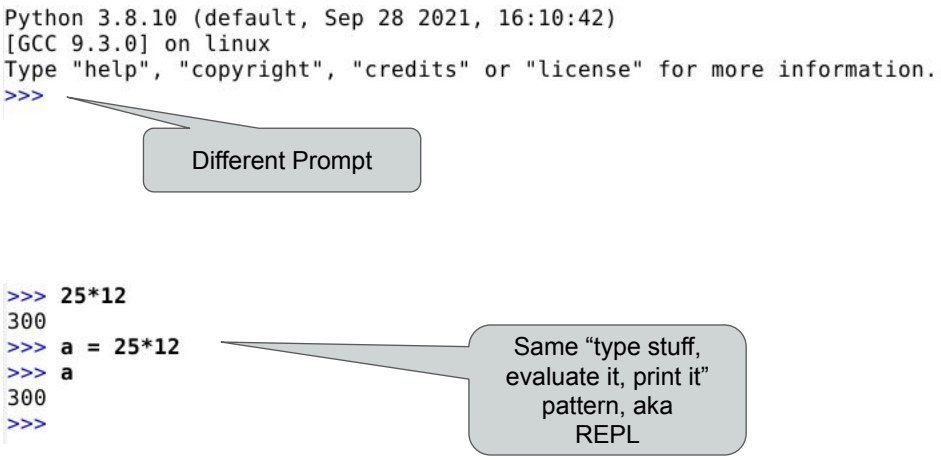
Python

CHICAS centre for health informatics, computing, and statistics

- Born 1991 - Guido van Rossum
- Open Source
- General Purpose
- Interpreted Language
- "There should be one - and preferably only one - obvious way to do it"

And if it wasn't for Guido van Rossum you might be learning perl now. Python was first released in 1991 as a new open source general purpose interpreted language with a bit of a programming philosophy behind it including this, which was the opposite of perl's stated philosophy of "There's More Than One Way To Do It" ("TMTOWTDI").

The Interpreter



When you run python interactively you get a message and a prompt which is three angle brackets. In this lecture if you see three angles its Python, if you see one its R. The interpreter works like R, you type an expression, Python evaluates it and prints the value. This is the “REPL” pattern of “Read - Evaluate - Print - Loop” which is common on many modern interpreted languages.

Similarities

- Numeric constants

```
>>> 23
23
>>> 98.4
98.4
>>> -1.234e-6
-1.234e-06
```

```
> 23
[1] 23
> 98.4
[1] 98.4
> -1.234e-6
[1] -1.234e-06
```

- Character constants

```
>>> "This"
'This'
>>> "Isn't R"
"Isn't R"
```

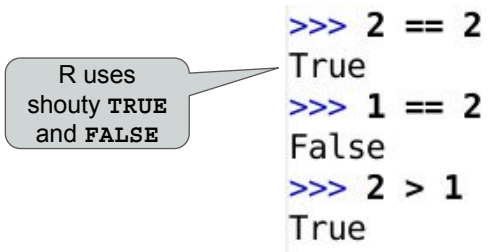
```
> "This"
[1] "This"
> "Isn't Python"
[1] "Isn't Python"
```

- All numerical expressions with `+ - * /` and `**`

And there's more commonalities with R. Numeric and character constants are all written the same way, and expressions with the basic four arithmetic operators and ** for "to the power of" are the same in Python as in R.

Similarities

Conditions with == > < <= >= !=



Conditions are the same too - equals, greater than etc. The only difference here being that R shouts TRUE at you and Python is more reserved.

Little Differences

CHICAS centre for health informatics, computing, and statistics

And?

```
> 2 > 1 & 3 > 2
[1] TRUE
```

```
>>> 2 > 1 & 3 > 2
False
```

This is one mistake I make with Python. In R, this looks right. 2 is greater than 1 and 3 is greater than 2. But in Python the same expression is False! How can this be false? I have done something like this a number of times in Python and I kick myself every time.

Little Differences

CHICAS centre for health informatics, computing, and statistics

And?

```
> 2 > 1 & 3 > 2
[1] TRUE
```

```
>>> 2 > 1 & 3 > 2
False
```

This is a binary bitwise-AND operator

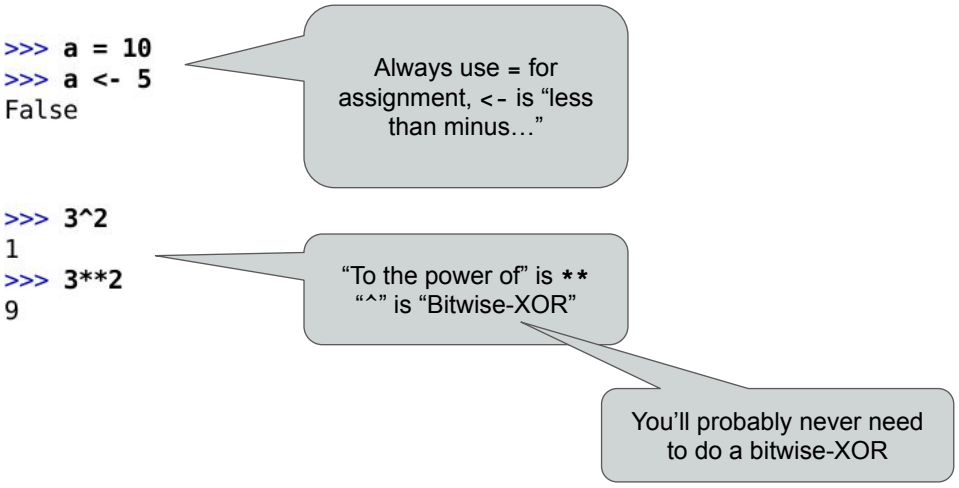
You will never need a bitwise-AND operator

```
>>> 2 > 1 and 3 > 2
True
```

Python uses `and` `or` and `not`

Python uses the word “and” to combine conditions! Not “&”! In python, that’s a binary operator which you’ll probably never need. Python uses “and”, “or” and “not” to combine comparison tests.

Little Differences



You might also note that python only uses the single equals sign for assignment, and any attempt to use the R-ish “<-” arrow is seen as an attempt to do “less than negative”. Another minor difference is in R using either of ** or ^ for “to the power of”, but in Python only ** does powers. The ^ operator does another binary operation which you probably never need. This is another source of “kick yourself, Barry” moments for me.

Difference

Triple-quotes define multi-line strings:

```
>>> message = """
... This is a message.
... It has more than one line.
...
... Message Ends.
... """
```

… is python's continuation prompt

```
>>> message
'\nThis is a message.\nIt has more than one line.\n\nMessage Ends.\n'
>>> print(message)

This is a message.
It has more than one line.

Message Ends.
```

String has newlines in it at line breaks

Python has a special way to create multi-line strings using triple-quotation mark "fences". This puts newline characters into a string so that when printed it comes out like it was in the source code. You can also see that python here prints three dots as a continuation prompt, much like R says "+".

Little Differences

R's vectors vs Python sequences

```
> primes = c(2,3,5,7,11,13)
> primes
[1]  2  3  5  7 11 13
```

R uses the `c()` function

```
>>> primes = [2,3,5,7,11,13]
>>> primes
[2, 3, 5, 7, 11, 13]
>>>
>>> primes = (2,3,5,7,11,13)
>>> primes
(2, 3, 5, 7, 11, 13)
```

Python uses square brackets

Or round brackets

There is a difference...

So lets look at data structures in the two languages. We can make a vector in R with the c(..) function. In Python we can make a list *sequence* using square brackets or a tuple *sequence* using round brackets. There's a subtle difference between them but most of the following will use list sequences and I'll probably call them both "lists" and "sequences". They're not *vectors* though.

Bigger Differences

CHICAS centre for health informatics, computing, and statistics

```
> primes * 2
[1]  4  6 10 14 22 26
```

With R, as a numerical-data-oriented language, operations tend to work on whole data sets. Multiply a vector by two and you get all the values multiplied by two.

Bigger Differences

CHICAS centre for health informatics, computing, and statistics

```
> primes * 2
[1]  4  6 10 14 22 26
```

```
>>> primes
[2, 3, 5, 7, 11, 13]
>>> primes * 2
[2, 3, 5, 7, 11, 13, 2, 3, 5, 7, 11, 13]
```

No implicit processing of sequence elements as in R

With python, multiplying a list by two gets you… double the list! Two copies of the list!

Python Sequences

Can store different types:

```
>>> a = ["This", 15, True]
>>> a
['This', 15, True]
```

Which makes the * 2 seem reasonable:

```
>>> a * 2
['This', 15, True, 'This', 15, True]
```

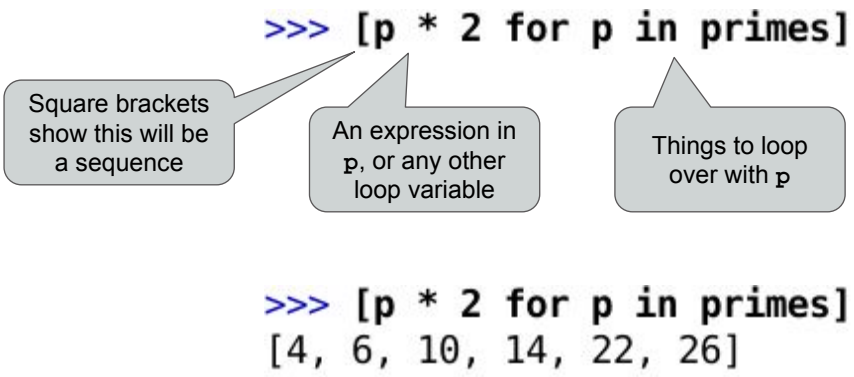
Any sequence can be doubled this way.

This makes sense for a more general programming language since R's sequences can store values of different types, and multiplying this by two doesn't make any sense. But any sequence can always be doubled by copying it...

Operating on Sequences

CHICAS centre for health informatics, computing, and statistics

List Comprehensions



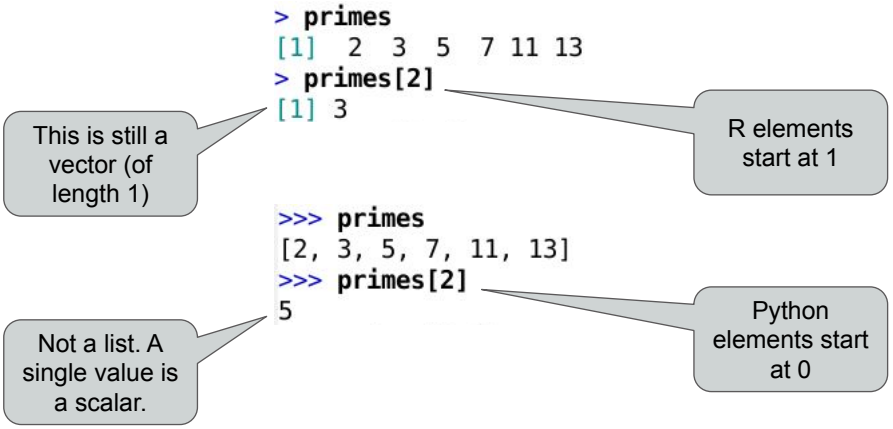

```
>>> [p * 2 for p in primes]
[4, 6, 10, 14, 22, 26]
```

If you do want to operate on each element of a sequence you have to be explicit, and a common way is using a *list comprehension*. This is an expression inside square brackets that loops a loop variable over the values of a sequence. What you get back is a list.

Big Differences

CHICAS centre for health informatics, computing, and statistics

Elements:



Now we can construct sequences, lets take them apart. You can get an element out of an R vector and the first element is element 1. In Python, elements are numbered from 0, so you get a different element here. Notice also that extracting a value from a Python sequence doesn't result in a sequence but a simple scalar (in this case) numeric value. There's no equivalent to this in R, even a number on its own is a vector of length 1.

Big Differences

CHICAS centre for health informatics, computing, and statistics

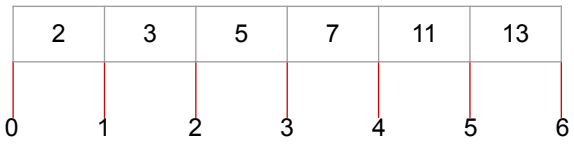
Subsets:

```
> primes[2:4]
[1] 3 5 7
```

Slices:

```
>>> primes[2:4]
[5, 7]
```

Think of these as ***the gaps between the elements***, not the elements themselves.



Slicing and subsetting is another difference that can catch people. In R, the syntax is of a subset, where the code in the brackets defines the elements you want to pick out. But in R, the meaning is a *slice", so you have to imagine cutting the list at those points. Think of the slice numbers as indexing the gaps between the elements, starting from zero, and not the elements themselves.

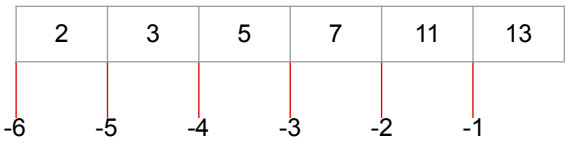
Big Difference - negative subsets

```
> primes
[1]  2  3  5  7 11 13
> primes[-3]
[1]  2  3  7 11 13
> primes[-2:-4]
[1]  2 11 13
> primes[-4:-2]
[1]  2 11 13
```

R, negative means “not this”

Python, negative means “index from end”

```
>>> primes[-4]
5
>>> primes[-4:-2]
[5, 7]
>>> primes[-4:]
[5, 7, 11, 13]
```



In R, a negative subset means “drop this element”, but it python it means “index from the end of the list”.

Big Difference - bounds

CHICAS centre for health informatics, computing, and statistics

R - NA

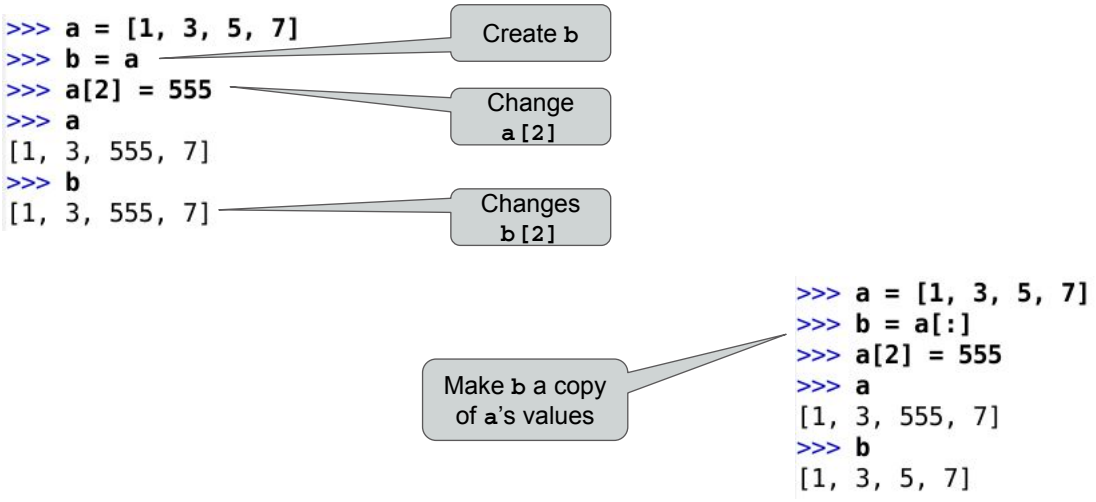
```
> primes
[1]  2  3  5  7 11 13
> primes[99]
[1] NA
```

Python: Nah!

```
>>> primes
[2, 3, 5, 7, 11, 13]
>>> primes[99]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

In R, if you access off the end of a vector you get an NA value. Perhaps this makes sense statistically speaking - if you have 10 data points then I suppose the 99th is NA “Not Available” - but most other languages will raise an error or fail silently (and possibly dangerously) in this situation. Python raises and error and stops execution.

Big Difference - copy-by-reference



Now for a major structural difference. If you copy a list by doing `a = b` in Python you are not copying the data - a and b still refer to the same data. So if you modify `a`, then `b` will change. This is unlike R where assignments like this create copies of the data as well. If you need a copy that you want to change without affecting the original, use a slice notation to do that.

Python sequences ~= R Lists

CHICAS centre for health informatics, computing, and statistics

Can hold other structures:

```
>>> student = ["Fred Smith", [ ["CHIC402", 73], ["CHIC602", 82] ] ]
```

```
>>> student[0]
'Fred Smith'
>>> student[1]
[['CHIC402', 73], ['CHIC602', 82]]
>>> len(student[1])
2
>>> student[1][0]
['CHIC402', 73]
```

Gets the name

Gets the list of courses

Number of courses

First of the courses

Python list sequences are a bit more like R lists in that they can store other structured objects - including lists inside lists. Here's a nested list, and the ways of accessing parts of it including the deeper levels of nesting.

Named elements

CHICAS centre for health informatics, computing, and statistics

In R, as a list with named elements:

```
student = list(
    name="Fred Smith",
    courses=list(
        list(course="CHIC402",mark=73),
        list(course="CHIC602",mark=82)
    )
)
```

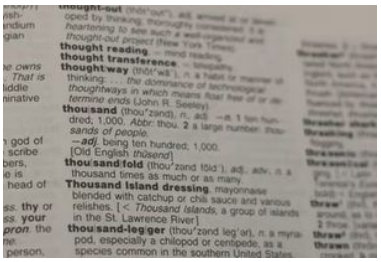
```
> student$courses[1]
[[1]]
[[1]]$course
[1] "CHIC402"

[[1]]$mark
[1] 73
```

R lets you give names to list elements, so that you can access components with meaningful names to make your code clearer.

Named elements

Python Dictionary

```
student = {
    'name': "Fred Smith",
    'courses': [
        {'name': "CHIC402", 'mark': 73},
        {'name': "CHIC602", 'mark': 82}
    ]
}
```

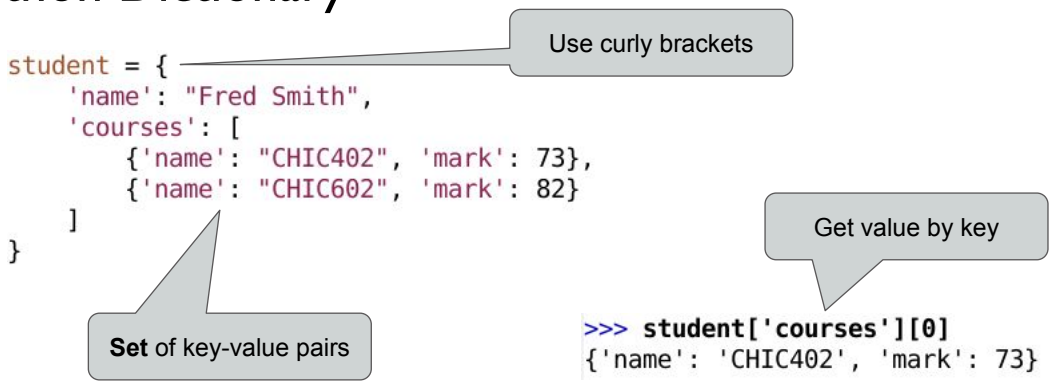


```
>>> student['courses'][0]
{'name': 'CHIC402', 'mark': 73}
```

In R, named access to elements is done using a “Dictionary”, because it provides a lookup of values like you look up a definition in a dictionary.

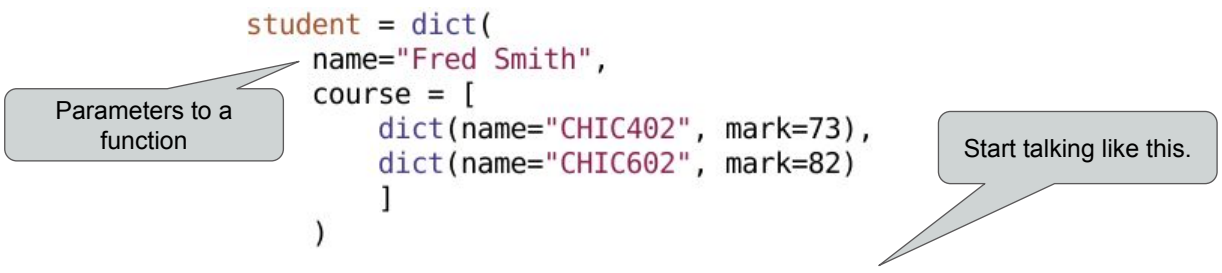
Named elements

Python Dictionary



You can make one in Python by constructing it with curly brackets and a set of key: value pairs like this. Here you can see the `courses` value is a list, and that list contains more dictionaries. To access an element of a dictionary, you look up via the key and get back the value.

Make a dictionary by function



"**`student`** is a dictionary where **`course`** is a list of course dictionaries and **`name`** is the student name"

"Course dictionaries have **`name`** and **`mark`** keys."

Another way of making a dictionary is by named arguments to the `dict` function. When you start building data structures like this its worth learning how to describe them in English - in this case we have a dictionary where "course" is a list of course dictionaries and "name" is the student name. A "Course" dictionary has keys "name" and "mark" giving the course name and exam score for the student.

A dict is a set

...and a set (mathematically defined) does not have an ordering...

```
>>> student[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 0
```

...and a key not in the dictionary is an error too...

```
>>> student['exams']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'exams'
```

Its important to note that a dictionary is a set in the mathematical sense, in that it has no inherent ordering, so getting an element by number will fail. Note also that if you look up something with a key that isn't in the dictionary this will fail also.

Difference

CHICAS centre for health informatics, computing, and statistics

R allows partial matching:

```
> student$name
[1] "Fred Smith"
> student$n
[1] "Fred Smith"
```

Python does not:

```
>>> student['name']
'Fred Smith'
>>> student['n']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'n'
```

Related to that, R is sloppy in its name matching in places, where Python is strict. You can't abbreviate dictionary keys or function argument names.

Difference: methods vs functions

CHICAS centre for health informatics, computing, and statistics

Python: methods

```
>>> z = [2, 3, 5, 7, 11, 13, 17]
>>> z.reverse()
>>> z
[17, 13, 11, 7, 5, 3, 2]
```

Can modify the object and/or return a value.

R: functions

```
> z = c(2,3,5,7,11,13,17)
> z = rev(z)
> z
[1] 17 13 11  7  5  3  2
```

Has to return new value

Here's another fundamental difference between Python and R. In R, the common paradigm is that a function returns a value so if you want to change an object you have to assign it a new value from the function call. But in Python there are "methods" that apply to objects, which are functions that can modify the object they are applied to. So a Python list has a set of available methods such as "reverse", which reverses the order of elements in the list.

Difference: methods vs functions

CHICAS centre for health informatics, computing, and statistics

Python: methods

```
>>> z = [2, 3, 5, 7, 11, 13, 17]
>>> z.reverse()
>>> z
[17, 13, 11, 7, 5, 3, 2]
>>> z.sort()
>>> z
[2, 3, 5, 7, 11, 13, 17]
>>> z.extend([19,23])
>>> z
[2, 3, 5, 7, 11, 13, 17, 19, 23]
```

R: functions

```
> z = c(2,3,5,7,11,13,17)
> z = rev(z)
> z
[1] 17 13 11  7  5  3  2
> z = sort(z)
> z
[1]  2  3  5  7 11 13 17
> z = c(z, c(19,23))
> z
[1]  2  3  5  7 11 13 17 19 23
```

Other elements include “sort” and “extend”. The equivalents in R are functions that need to be assigned back to the object if you wish to replace the value of the object, but in Python the method changes the value of the object in place.

Methods return a value

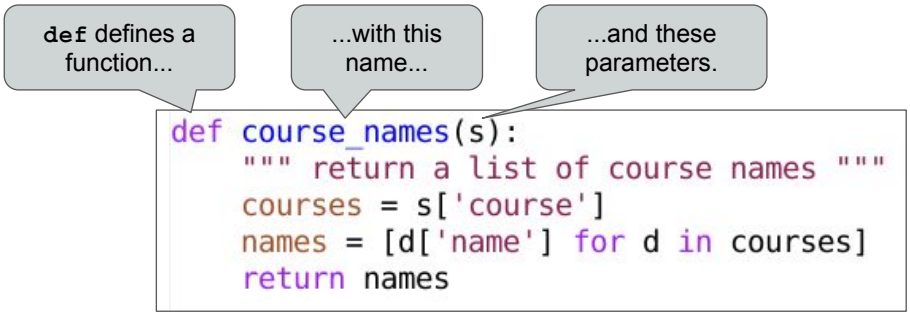
CHICAS centre for health informatics, computing, and statistics

Count of values:

```
>>> N = [1,2,6,5,5,4,5,2,3]
>>> N.count(5)
3
>>> N.count(99)
0
```

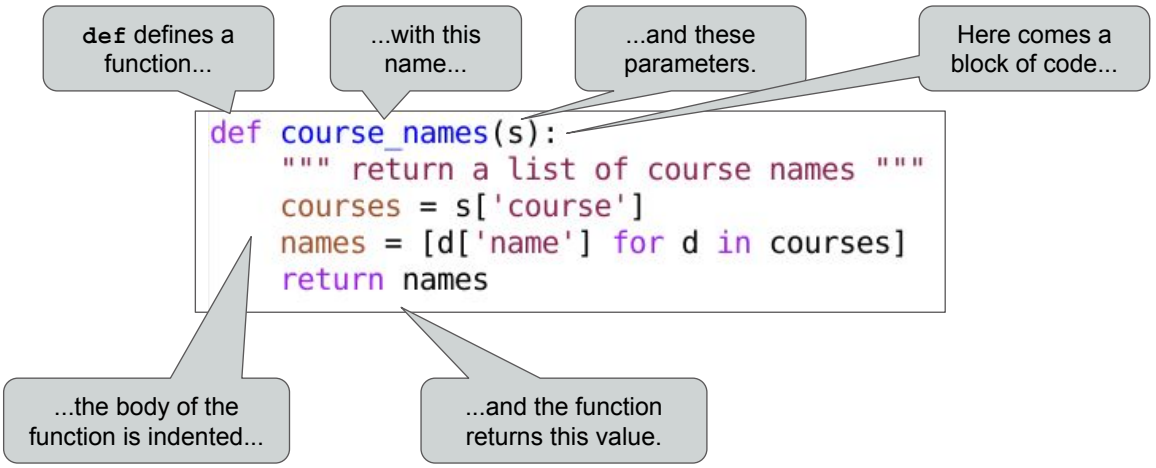
Methods can also return a value (without changing the object) such as the “count” method which returns the number of occurrences of a value in a sequence.

Functions



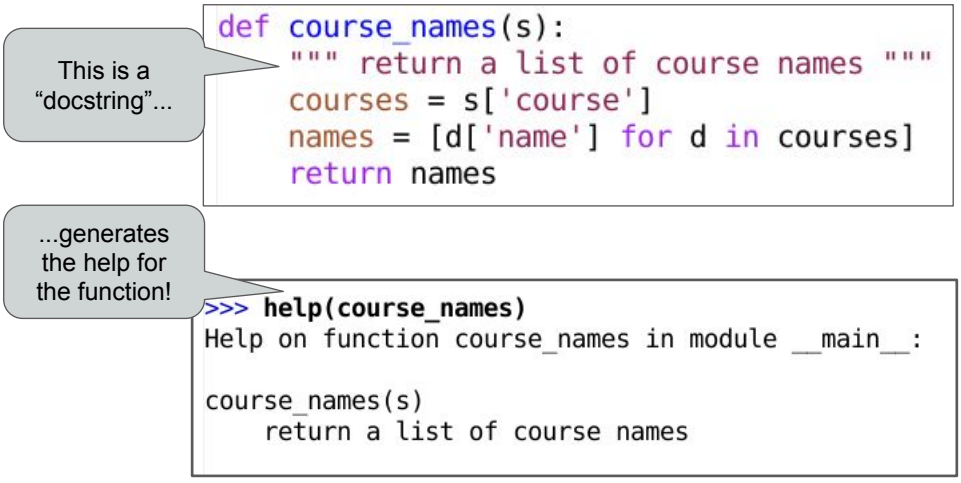
Python does also have functions and you can create your own with `def`. Choose a name and a set of parameters.

Functions



The colon after the argument list introduces the body of the function code. This is indented, and ends when the indenting does (perhaps a new function is defined) or at the end of the file. The return statement specifies the value for the function to return at that point.

Functions



That triple-quoted string just after the definition is a special comment called a "docstring" or "documentation string". Its an optional extra that will appear if anyone runs the help function on your function. This is a good thing and worth doing to all your functions.

Python Help

CHICAS centre for health informatics, computing, and statistics

Built-in, also can appear in IDE help popups:

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

The help system is quite extensive.

Comparison with R

CHICAS centre for health informatics, computing, and statistics

```python
def course_names(s):
    """ return a list of course names """
    courses = s['course']
    names = [d['name'] for d in courses]
    return names
```

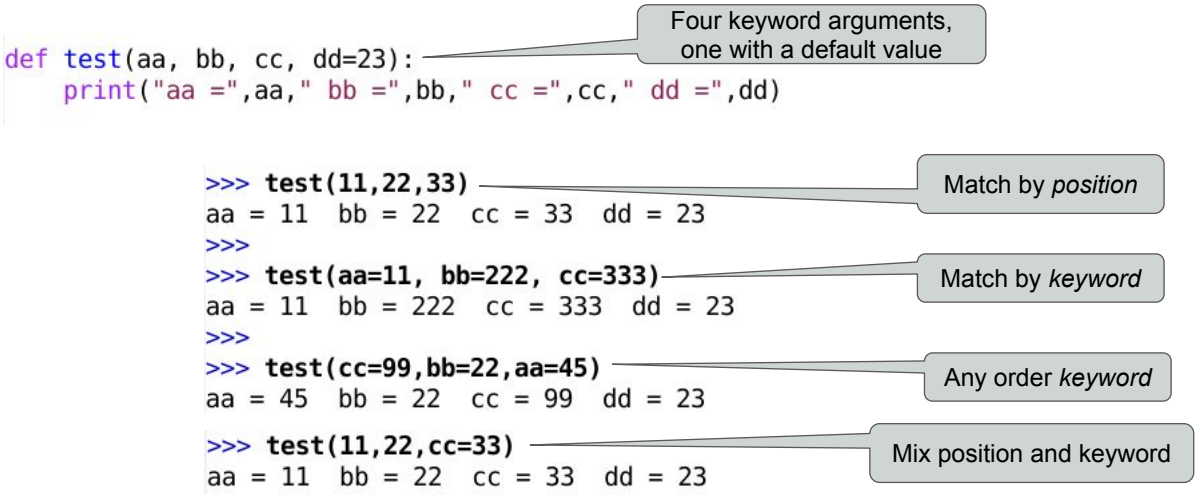
- Indentation defines body
- Needs **`return`**
- Exact parameter matches only
- docstring

```r
course_names <- function(s){
### return a vector of course names
    courses = s$course
    sapply(courses, function(c){c$course})
}
```

- Curly brackets define body
- Last value returns
- Partial parameter matches
- Use comments

Here's a side-by-side python-R comparison. In Python, indentation defines the body of the function, in R its within curly brackets. In Python you need the return statement, in R you can let the code fall through and it returns the last evaluated expression. In Python you have to exactly match parameter names, in R these can be partial matches. In Python you can use a docstring and comments in the code (with a # sign) and in R you use # for comments and there are complex ways to build the help pages.

Argument name matching



Python and R have similar ways to match the passed parameters to the function definition. Python can have arguments matched by position or keyword. Usually what you use is the option that makes your usage clearest, and some of the ways of mixing them up are a bit perverse and not recommended.

Dont…

```
>>> test(cc=99,22,aa=45)
  File "<stdin>", line 1
SyntaxError: positional argument follows keyword argument
>>> test(aa=1,b=2,c=3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: test() got an unexpected keyword argument 'b'
>>> test(11,22,aa=33)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: test() got multiple values for argument 'aa'
```

Confusing….

Must be exact matches

First is `aa`, so what's this third one?

Some combinations are ambiguous so Python will throw an error. Some of these work in R but you have to know R's procedure for matching. The best thing to do is to avoid anything that looks a bit weird.

If you want to return...

CHICAS centre for health informatics, computing, and statistics

Have to use the **`return`** statement:

```
def no_return(a):
    a * 2

>>> b = no_return(123)
>>> b
>>>
```

```
def no_return(a):
    return a * 2

>>> b = no_return(123)
>>> b
246
```

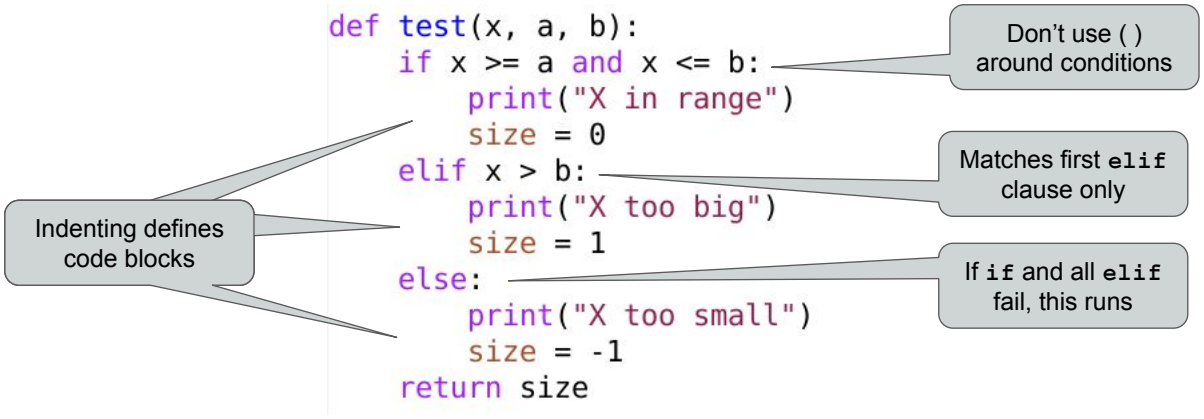
Note its not a function:

```
def no_return(a):
    return(a * 2)
```

works but is subtly different...

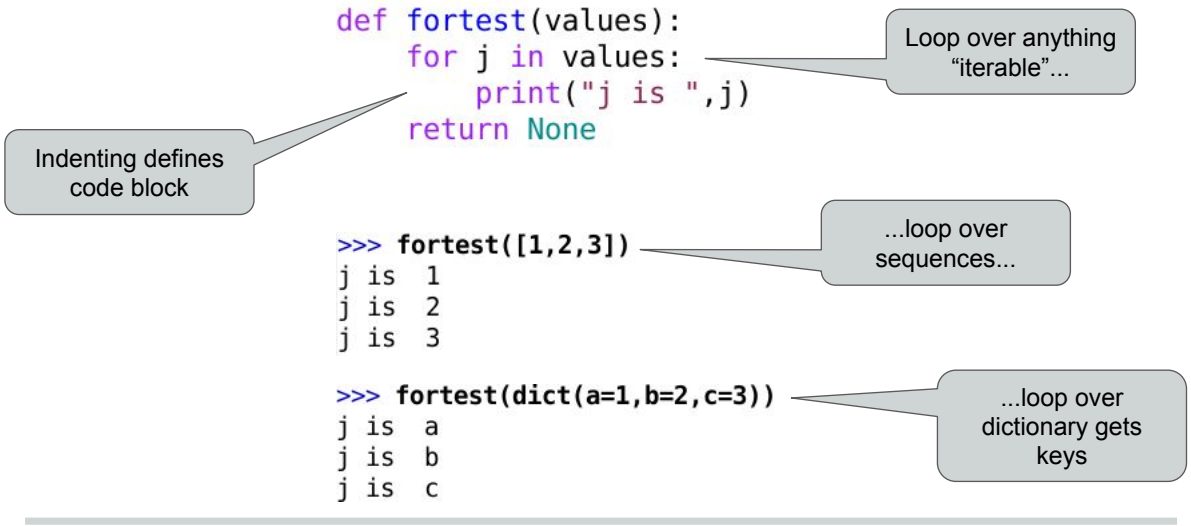
As I mentioned, Python requires the `return` statement in order for a function to return a value, otherwise it returns “None”. Note that its not a function, so you don’t need to call it like a function in R. This does work but its best avoided.

Control Structures: if



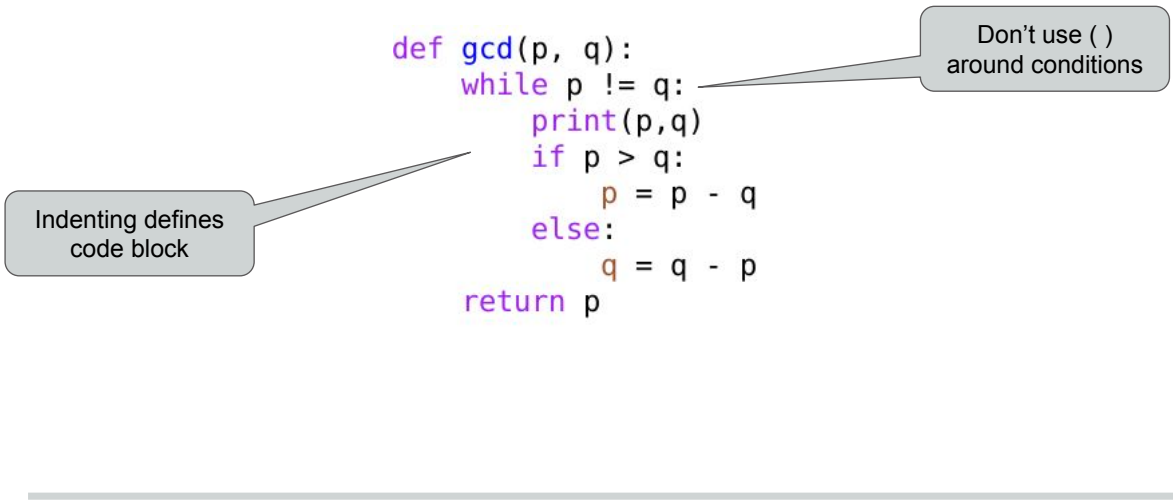
Python's control structures are very similar to R except there's no curly brackets. The blocks are again defined by whitespace indenting. Also there's no parentheses around the test conditions and there's the : that introduces a new code block. You can have one `if` block, zero or more `elif` blocks, and zero or one `else` block. Only one of the `if` or `elif` blocks will be executed, and if none of them are then the `else` block runs.

Control Structures: for



Python's looping controls include a `for` statement that runs a block of code with a loop variable set to values of a sequence. This is actually extended to anything Python can see as "iterable", meaning "it can be iterated over" and the results of iterating over different things produces different results. For example iterating over a dictionary produces a loop over the keys of the dictionary.

Control Structures: while



There's also the `while` loop. Similar syntax - a condition followed by a colon introduces an indented block which repeats until the condition is True.

Modules

CHICAS centre for health informatics, computing, and statistics

arithmetic.py

```
def gcd(p, q):
    while p != q:
        print(p,q)
        if p > q:
            p = p - q
        else:
            q = q - p
    return p
```

If I have a file called **arithmetic.py**...

...I can *import* it and use functions defined in it...

```
>>> import arithmetic
>>> G = arithmetic.gcd(9, 12)
9 12
9 3
6 3
>>> G
3
```

Python lends itself to writing modular code so you can spread useful functions across different files for re-use. If I have a function definition in a file, I can use the `import` statement to add that module to my session, and then call functions in it by using the dot-notation. I can put many related functions into a single file and call them all from `arithmetic` if I want.

The Standard Library

CHICAS centre for health informatics, computing, and statistics

https://docs.python.org/3/library/index.html

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.factorial(10)
3628800
>>> math.gcd(9,12)
3
>>> math.sqrt(math.factorial(10))
1904.9409439665053
```

And this is how the standard python library works. This is a large body of modules that are distributed with 99.9% of all Python installations. For example, there's the `math` module with assorted mathematical functions and constants.

Help

```
>>> help(math)
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module provides access to the mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of

    asinh(x, /)
        Return the inverse hyperbolic sine of x.
```

```
>>> help(math.sin)
Help on built-in function sin in module math:

sin(x, /)
    Return the sine of x (measured in radians).
```

If you want to find out what's in a module, look at its `help` text. Or you can look at a function's help as well.

Installation and Setup

CHICAS centre for health informatics, computing, and statistics

- System Python **www.python.org**
  - Installs Python on your PC
- "Thonny" **www.thonny.org**
  - Simple IDE bundled with a Python
- Conda, Anaconda, Miniconda
  - Include an add-on package management system
- Docker
  - Lets you set up "containers", mini-virtual machines that run isolated from each other.

Installing python can vary from easy to annoying. Its not helped by the multiple ways of getting it on your system and the various operating systems and configurations of machines. The "standard" way is to install from www.python.org and that's recommended for Windows which doesn't have a "system" python on it already. Macs and Linux boxes will usually have python installed with the base operating system (since it is used by the OS for lots of things) and installing a new one from www.python.org can have bad effects. In this instance, you might be better off installing using Conda: https://docs.conda.io/en/latest/ which provides an environment for a python interpreter and an add-on package manager that doesn't interfere with a system python. There's also "Thonny", which is a simple editor and interpreter window for Python, with a small install footprint yet with everything bundled in (including Python) and might be worth trying if the system python or conda options are failing for you. You might also see people using `docker` to run python, which is a way of running "containers" on you system. A "Docker Container" is like a virtual PC on your system that has an isolated operating system from your native operating system.

Installation and Setup

CHICAS centre for health informatics, computing, and statistics

- System Python **www.python.org**
  - Installs Python on your PC
- "Thonny" **www.thonny.org**
  - Simple IDE bundled with a Python
- Conda, Anaconda, Miniconda
  - Include an add-on package management system
- Docker
  - Lets you set up "containers", mini-virtual machines that run isolated from each other.

I would recommend for now you try installing a system python from python.org, especially if you are running Windows, otherwise give Thonny a go (Windows/Mac/Linux) and otherwise Conda.

Interfaces

CHICAS centre for health informatics, computing, and statistics

Python

IPython

JupyterLab

VS Code

RStudio

IDLE

The other complication is the range of interfaces, from the simple Python command line through to advanced interfaces and development environments.

Interfaces

CHICAS centre for health informatics, computing, and statistics

Python

IPython

JupyterLab

VS Code

RStudio

IDLE

This is the plain Python terminal interface. Its uncluttered, low-featured, but it starts quick and does the job.

Interfaces

CHICAS centre for health informatics, computing, and statistics

Python

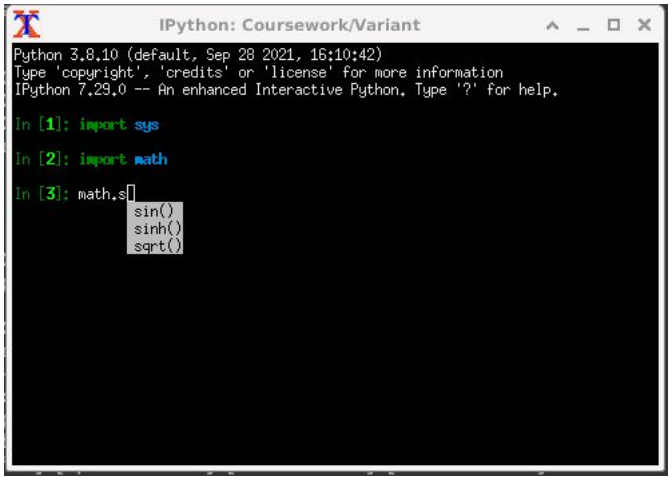
IPython

JupyterLab

VS Code

RStudio

IDLE



The IPython interface is designed with interactive work in mind, and has tab-completion and colours elements nicely.

Interfaces

Python

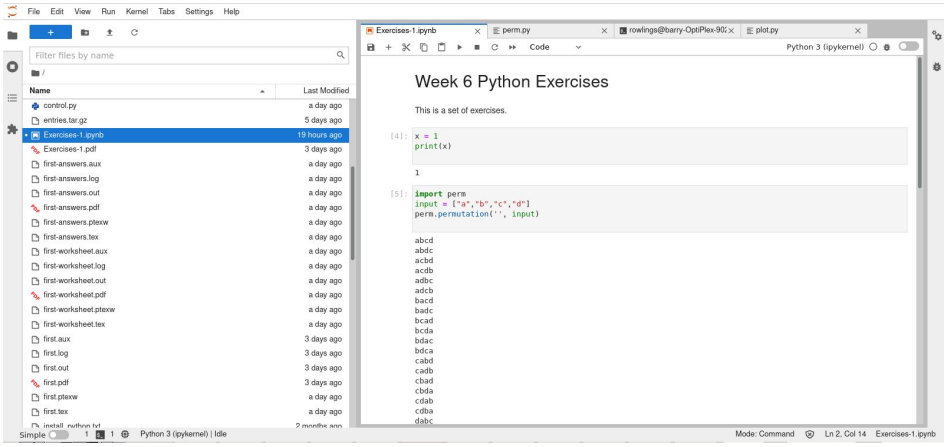
IPython

JupyterLab

VS Code

RStudio

IDLE



JupyterLab is an advanced web-based system that includes a file editor and a notebook editor.

Interfaces

Python

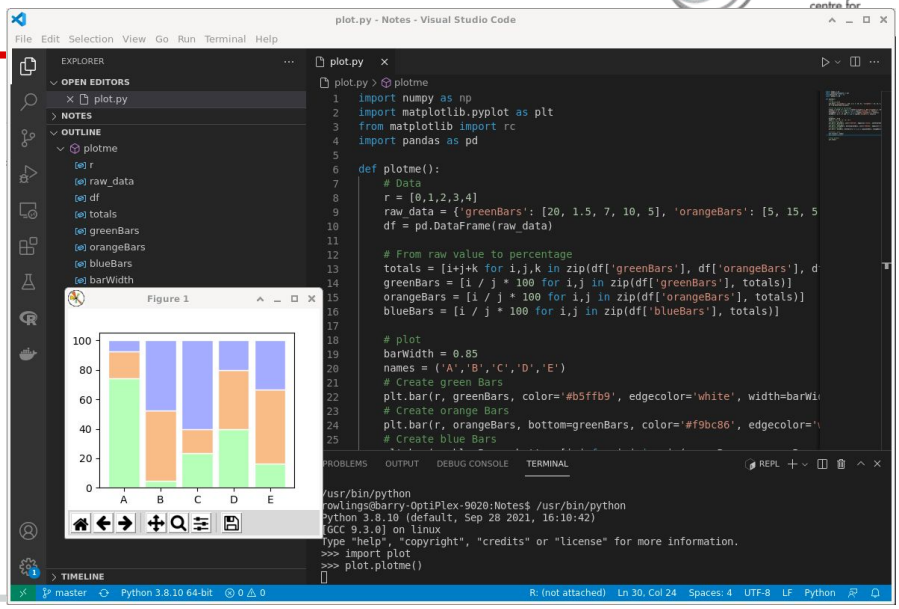
IPython

JupyterLab

VS Code

RStudio

IDLE



Microsoft VS Code also supports Python, which you can run interactively alongside all the other features of VS Code.

Interfaces

Python

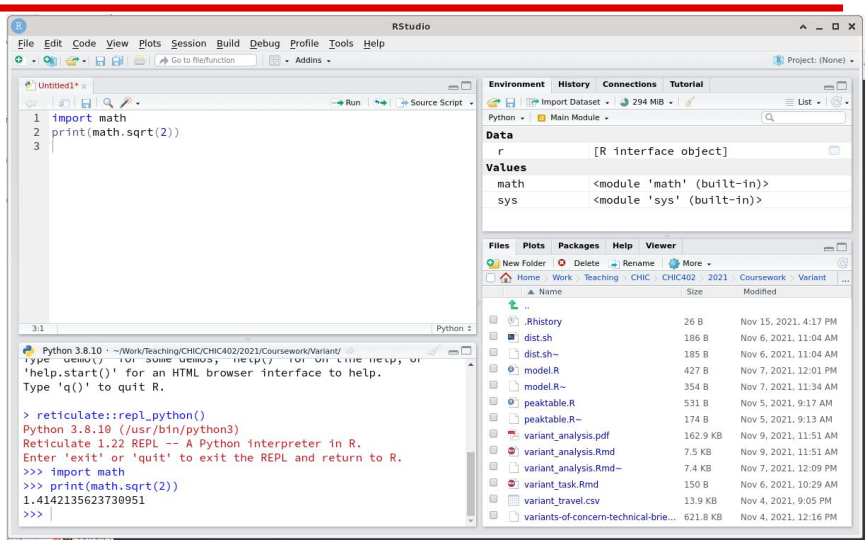
IPython

JupyterLab

VS Code

RStudio

IDLE



RStudio has some python support via the `reticulate` package, which you need to install first.

Interfaces

Python

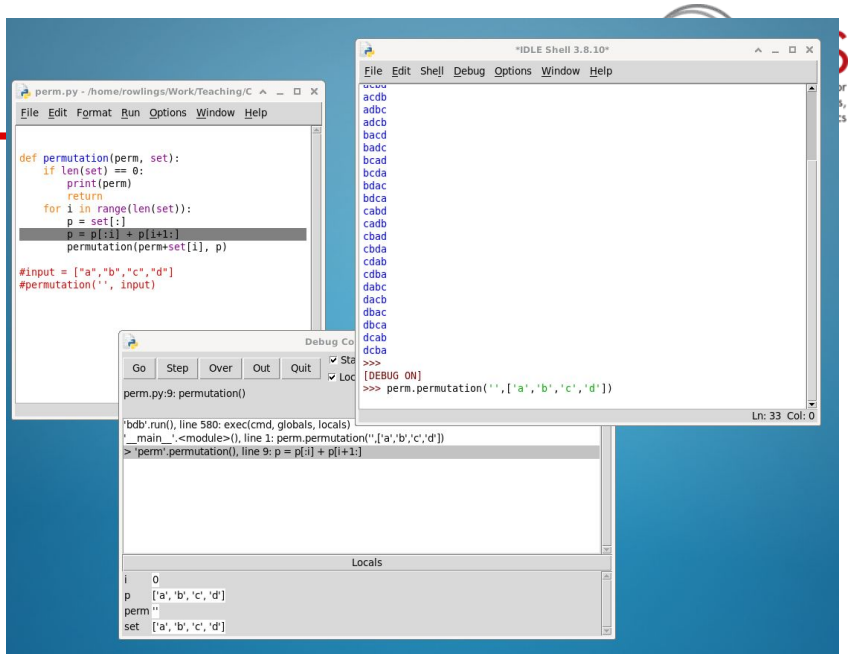
IPython

JupyterLab

VS Code

RStudio

IDLE



And then there's the earlier generation of development environments such as IDLE, which is bundled with some Python distributions and provides a multi-window interface with editors and interpreters.

Interfaces

Python

IPython

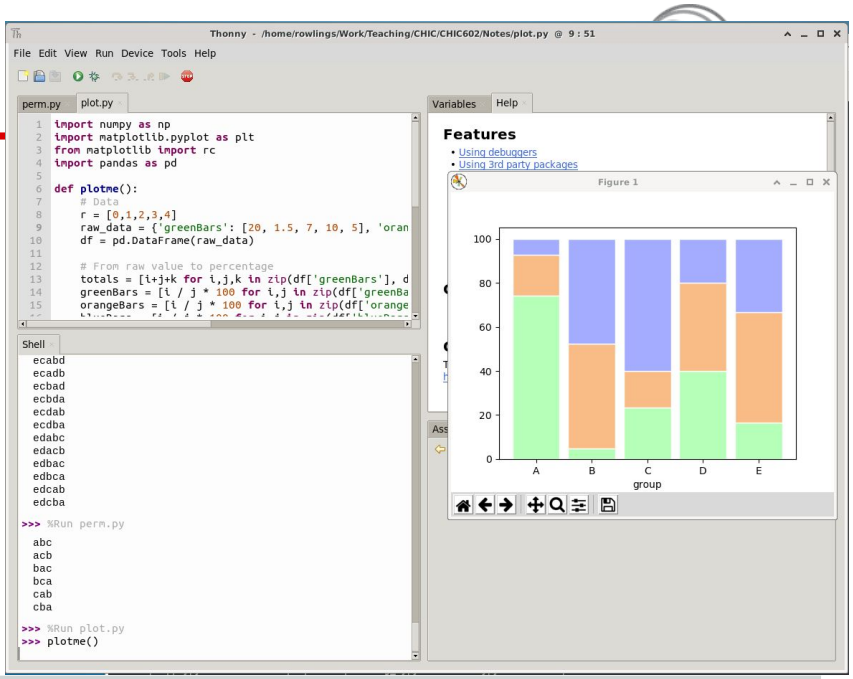
JupyterLab

VS Code

RStudio

IDLE

Thonny



Added to this list is Thonny, which is a single-window multi-pane interface which comes with a complete python installation in a single bundle for easy setup.

Virtual Environments

CHICAS centre for health informatics, computing, and statistics

- Wraps:
  - A Python Version
  - A Module Installer
  - A Set of Modules
- Allows:
  - Export of current module set (with versions)
  - Reproduction of environments
  - Isolated environments that don't clash.

There's also the complication of the "Python Virtual Environment" or "venv". Once you start using Python for different projects you'll find each project needs a different set of addon modules. Virtual Environments enable each project to have its own Python interpreter and set of installed packages. The list of installed packages can be written out to a "requirements" file so the environment can be replicated elsewhere.

Python

CHICAS centre for health informatics, computing, and statistics

- Its a bit like R.
- The bits not like R will break your code.
- There's more to come!
  - Classes and Objects
  - Numeric Python
  - Scientific Python
  - Graphics

So that's Python. Its a bit like R, except where it isn't, and those points can break your code (and sometimes you…). Coming up, more on Python's classes and objects, then the numeric and scientific python module stack, and graphics and charts….