Welcome to The Carpentries Etherpad!

This pad is synchronized as you type, so that everyone viewing this page sees the same text. This allows you to collaborate seamlessly on documents.

Use of this service is restricted to members of The Carpentries community; this is not for general purpose use (for that, try https://etherpad.wikimedia.org).

Users are expected to follow our code of conduct: https://docs.carpentries.org/topic_folders/policies/code-of-conduct.html



---

# Day 1: Spreadsheets and OpenRefine

- **10:00-10:10 Intro**
- **10:10 -11:00 Data organisation with Spreadsheets (Andrzej)**
- **11:00-11:10 Coffee break 1**
- **11:10-12:15 Data organisation with Spreadsheets (Andrzej)**
- **12:15-13:15 Lunch break**
- **13:15-14:00 Data cleaning with OpenRefine (Andrzej)**
- **14:00-14:15 Coffee break 2**
- **14:15-15:00 Data cleaning with OpenRefine (Andrzej)**

**Before you start (Spreadsheets, https://datacarpentry.org/spreadsheets-socialsci/):**

*Make sure you have a spreadsheet program installed before the workshop (Excel, Apple Numbers etc.). A free spreadsheet editor (part of the Libre Office open-source project) can be downloaded here: https://www.libreoffice.org/download/download/
For installation instructions on different operating software, see this page:

https://datacarpentry.org/spreadsheets-socialsci/setup.html

*Download all necessary files:
SAFI_clean.csv - https://ndownloader.figshare.com/files/11492171
SAFI_messy.xlsx - https://ndownloader.figshare.com/files/11502824
SAFI_dates.xlsx - https://ndownloader.figshare.com/files/11502827

*Make sure you have all the files in the same, accesible folder

Presentation used today can be downloaded here:
https://docs.google.com/presentation/d/1lKEVtGBxJFwxAlTfJsIbFVnKBh_b5UOj/edit?usp=sharing&ouid=104347110123739783728&rtpof=true&sd=true

**Before you start (OpenRefine, https://datacarpentry.org/openrefine-socialsci/):**

*Download OpenRefine package from: https://openrefine.org/download.html
On Windows, you simply unpack the zipped folder. See installation instructions here:
https://datacarpentry.org/openrefine-socialsci/setup.html

* Download the file we will work on here: https://ndownloader.figshare.com/files/11502815

*Best is to have both OpenRefine and the data file in the same folder

**Most common issue:**

The most common issue encountered is a lack of administrative privileges. If you plan to access the workshop from your study/workplace computer, firstly make sure you have the relevant software preinstalled or can install it on your own. If not, you will have to ask a relevant person in charge (IT/system administrator) to install and prepare software for you. It should be simpler for home computers, with OpenRefine sometimes requiring to open the exectutable file as administrator (on Windows, can be done by right click -> run as administrator, for one time only, or by navigating to properties -> comparability -> run as admin -> apply, to set it permanently).

Code of Conduct: https://docs.carpentries.org/topic_folders/policies/code-of-conduct.html

## Attendance, Day 1 (write your name):

Andrzej Romaniuk (instructor, day 1) -> https://andrzejromaniuk.github.io/CV/
Lucie Woellenstein (host/helper)
Athanasia Yiapanas (Helper)
Emma Hobbs (Helper)
Lorraine Linton
Sharon McMeekin
Sofiah MacLeod
Nicola Raine
Ivet Gazova
Sarah Knowles
Iain Scherr

Kenneth (Ken) Humphreys
Jennifer Ozers
Carol Campbell
Siddharth Manohar
Monica Mastrantonio

## EXERCISE 1, ANSWERS FROM EACH GROUP:

Room 5:

Issues with data entry - Mozambique

- Unclear if key_id is consistent across table (2 have 10 entries, one has 9)
- Typos/differences in data entry
- Notes added through use of highlighting - comp not able to pick up on this
- Why 3 tables? An issue for a computer processing it
- Obviously wrong number entered (-99 and -999)
- Concatenating of data in 0table
- In plots table under water use there is wild inconsistency in data entry, yes/no, Y/N, entries with notes, numbers
- Gaps - are they null values, zeroes or have they been accidentally left out?

Issues with data entry - Tanzania

- Unclear if key_id is consistent across table (1 has 10 entries, one has 9)
- Notes added through use of asterisks - comp not able to pick up on this
- Not the same data captured as with Mozambique, nothing on plots
- One table includes the title, one does not
- Has separated livestock data, but then entered inconsistently (mix of number and yes/no, also not consistently recorded 0s)
- Include totals, necessary in raw data? especially as they don't match the data in the rows in all places

Lucie's Room

- Spelling mistakes
- No uniform categories
- Highlighting loses information if exported as a csv file
- 3 tables in one spreadsheet - one table might be more useful
- Not all information in mozambique was collected for Tanzania
- Requires deductive reasoning to understand - python would not
- -99 what does that mean?
- Uniform way to say yes or no
- How to record missing
- Not the same number of key ids each table
- everything open to interpretation

Athanasia's Room

- Information in highlighting cannot be read for analysis
- inconsistency in use of underscore in variable names (2 words = 2 variables in some analysis software)
- Data duplication
- More than one information type in single field
- Numbers with asterix won't be recognized for analysis
- Would key_id ordering with similar items grouped help (e.g. each number of rooms for same dwelling type listed consecutively)
- Footnote information lost if export data

## EXERCISE 2, ANSWERS FROM EACH GROUP:

**What kind of metadata?**

- Each variable explained what it is, what format it is in? Is it numeric? String?
- what metrics are used?
- Date - what format? what is the interview date? What does it mean? IS the time necessary? Why the time?
- who owns the data?
- is it open access? under what license?
- Who to contact with questions?

## LINK FOR TODAY'S SPREADSHEET PRESENTATION:

https://docs.google.com/presentation/d/1FshrjZjM5d1CP0zZR81QDcBfqp7GsrhI/edit?usp=sharing&ouid=104347110123739783728&rtpof=true&sd=true

## INSTRUCTOR"S EXAMPLE OF A PROPERLY FORMATTED DATASET (EXERCISE 1):

https://docs.google.com/spreadsheets/d/1oHa9rsapZRd07X244oolHuqAQg4mL5pN/edit?usp=sharing&ouid=104347110123739783728&rtpof=true&sd=true

## OPENREFINE, EXERCISES ANSWER:

Exercise for editing the cell items_owned
Click edit cells -> transform on the down arrow for items_owned
value.replace("[","").replace("]","").replace("","").replace(" ","")
Click facet -> custom text facet on the down arrow for items_owned
value.split(";")

**REPLICABLE CODE:**


```
[
  {
    "op": "core/mass-edit",
    "engineConfig": {
      "facets": [],
      "mode": "row-based"
    },
    "columnName": "village",
    "expression": "value",
    "edits": [
      {
        "from": [
          "Ruaca",
          "Ruca"
        ],
        "fromBlank": false,
        "fromError": false,
        "to": "Ruaca"
      },
      {
        "from": [
          "Ruaca-Nhamuenda",
          "Ruaca - Nhamuenda"
        ],
        "fromBlank": false,
        "fromError": false,
        "to": "Ruaca-Nhamuenda"
      }
    ],
    "description": "Mass edit cells in column village"
  },
  {
    "op": "core/mass-edit",
    "engineConfig": {
      "facets": [],
      "mode": "row-based"
    },
    "columnName": "village",
    "expression": "value",
    "edits": [
      {
        "from": [
          "Chirdozo"
        ],
        "fromBlank": false,
        "fromError": false,
        "to": "Chirodzo"
```

```
    }
  ],
  "description": "Mass edit cells in column village"
},
{
  "op": "core/mass-edit",
  "engineConfig": {
    "facets": [],
    "mode": "row-based"
  },
  "columnName": "village",
  "expression": "value",
  "edits": [
    {
      "from": [
        "Ruaca-Nhamuenda"
      ],
      "fromBlank": false,
      "fromError": false,
      "to": "Ruaca"
    }
  ],
  "description": "Mass edit cells in column village"
},
{
  "op": "core/text-transform",
  "engineConfig": {
    "facets": [],
    "mode": "row-based"
  },
  "columnName": "interview_date",
  "expression": "value.toDate()",
  "onError": "keep-original",
  "repeat": false,
  "repeatCount": 10,
  "description": "Text transform on cells in column interview_date using expression value.toDate()"
},
{
  "op": "core/text-transform",
  "engineConfig": {
    "facets": [],
    "mode": "row-based"
  },
  "columnName": "items_owned",
  "expression": "grel:value.replace(\"[\",\"\").replace(\"]\",\"\").replace(\"\"\",\"\").replace(\" \",\"\")",
  "onError": "keep-original",
  "repeat": false,
  "repeatCount": 10,
  "description": "Text transform on cells in column items_owned using expression
```

```
grel:value.replace(\"[\",\"\").replace(\"]\",\"\").replace(\"\"\",\"\").replace(\" \",\"\")"
  },
  {
    "op": "core/text-transform",
    "engineConfig": {
      "facets": [],
      "mode": "row-based"
    },
    "columnName": "months_lack_food",
    "expression": "grel:value.replace(\"[\",\"\").replace(\"]\",\"\").replace(\"\"\",\"\").replace(\" \",\"\")",
    "onError": "keep-original",
    "repeat": false,
    "repeatCount": 10,
    "description": "Text transform on cells in column months_lack_food using expression
grel:value.replace(\"[\",\"\").replace(\"]\",\"\").replace(\"\"\",\"\").replace(\" \",\"\")"
  },
  {
    "op": "core/text-transform",
    "engineConfig": {
      "facets": [],
      "mode": "row-based"
    },
    "columnName": "village",
    "expression": "value.trim()",
    "onError": "keep-original",
    "repeat": false,
    "repeatCount": 10,
    "description": "Text transform on cells in column village using expression value.trim()"
  },
  {
    "op": "core/text-transform",
    "engineConfig": {
      "facets": [],
      "mode": "row-based"
    },
    "columnName": "years_farm",
    "expression": "value.toNumber()",
    "onError": "keep-original",
    "repeat": false,
    "repeatCount": 10,
    "description": "Text transform on cells in column years_farm using expression value.toNumber()"
  },
  {
    "op": "core/text-transform",
    "engineConfig": {
      "facets": [],
      "mode": "row-based"
    },
    "columnName": "years_farm",
```

```
    "expression": "value.toNumber()",
    "onError": "keep-original",
    "repeat": false,
    "repeatCount": 10,
    "description": "Text transform on cells in column years_farm using expression value.toNumber()"
  },
  {
    "op": "core/mass-edit",
    "engineConfig": {
      "facets": [],
      "mode": "row-based"
    },
    "columnName": "village",
    "expression": "value",
    "edits": [
      {
        "from": [
          "49"
        ],
        "fromBlank": false,
        "fromError": false,
        "to": "Chirodzo"
      }
    ],
    "description": "Mass edit cells in column village"
  }
]
```


# Day 2: Python

## Time schedule Wednesday

- **10:00-10:15 Intro**
- **10:15 -11:00 Python (Francesco)**
- **11:00-11:15 Coffee break 1**
- **11:15-12:00 Python (Francesco)**
- **12:00-13:00 Lunch break**
- **13:00-14:00 Python (Francesco)**
- **14:00-14:15 Coffee break 2**
- **14:15-15:00 Python (Francesco)**

## Before you start

**(Python https://librarycarpentry.org/lc-python-intro/ )**

Make sure you have installed Python eg Anaconda.
We are using the **Spyder** application that comes with Anaconda though you can use Jupyter.

## Attendance, Day 2 (write your name):

Francesco Tabaro (instructor)
Gina (host)
Phil Reed (helper)
Silvia Shen (helper)
Ken Humphreys (attendee)
Siddharth Manohar
Sofiah MacLeod
Ivet Gazova
Anna Mazurek
Nicola Raine
Iain Scherr
Lorraine Linton
Jennifer Ozers
Sharon McMeekin
Sarah Knowles
Monica Mastrantonio
Carol Campbell

# What Python software are you using?

Some Anaconda, some Spyder. We are looking at **Spyder**.
It has a panel for typing scripts, a panel to browse files, and a panel for the console to run commands.

# The code from today

Variables (number or string) characters are wrapped in quotes

```
age = 42
first_name = 'Francesco'
```

Invalid variable names: 1st_name, first name, first-name

Call a function to print the variable content:

```
print(first_name, 'is', age, 'years old.')
```

Do some maths

```
age = age + 3
print(age)
```

Use indices to select characters within a string, count from zero

```
print(first_name[0])
```

Use slices to select portions of strings

```
print(first_name[0:3])
```

How many letters long? len function

```
print(len(first_name))
```

Names are case sensitive

```
First_name = 'Anna'
print(first_name)
```

## Exercise

What is happening here?

```
x = 1.0
y = 3.0
swap = x     # assign the value of x into the variable swap (1.0)
x = y
y = swap     # need a third variable if you want to swap two variables
```

# What is the value of the potition variable?

```
initial = 'left'
potition = initial
initial = 'right'
```

# What is the value of the position variable now?

Slices count from the first position to just before the last position

```
library_name = 'social sciences'
print('library_name[1:3] is:', library_name[1:3])
# oc
```

Leave out the last position to run all the way to the end

```
print(library_name[7:])
# sciences

print(library_name[:])
# leave out start and end to get the whole string
```

Start counting backwards from the end

```
print(library_name[0:-1])
```

What happens if you print a variable that does not exist?

```
print(library_cat)
# error, you need to define library_cat before you use it
```

What happens if you use an index which does not exist?

Python can tell us what type of variable you have.

```
type(age)
# int
type(first_name)
# str
```

We can do algebra on numbers. If we try with string, we get a type error.

```
age = age + 3
first_name + 3
# error
```

But... if we use + on strings, they join up (concatenate)

```
first_name = first_name + '!'
'Francesco' + ' ' + 'Tabaro'
```

You can do multiplication on strings

```
'=' * 10
# '=========='
```

Combine with length function

```
ten_equals = '=' *  10
print(len(ten_equals))
```

# 10

You can convert types.
Would this work? 1 + '2'  No
Need to write this way:

```
1 + int('2')
str(1) + '2'
```

These give different results. 3 and '12'

You can mix integer (whole) and floating point (decimal) numbers

```
print (1 / 2.0)
# 0.5
type(1)         # int
type(2.0)       # float
type(1 / 2.0)   # float

print(2 ** 3)   # 8
print(2.0 ** 3)  # 8.0

type(3.4)       # float
result = 3.25 + 4
type(result)    # float
```

If there is a float in the line, the result becomes a float

## Exercise

What kind of variables would we use for...

- Number of days since the start of the year: int
- Time elapsed since the start of the year: float
- Call number of a book: string (can also have letters)

## Coffee break, continue with the lesson

Python has some different types of division, use different symbols

- Floating point: 5 / 3 = 1.666666667
- Floor: 5 // 3 = 1  (rounding down)
- Modulo: 5 % 3 = 2 (the remainder)

## Exercise

```
num_students = 600
num_per_class = 42
```

```
# how many students do we need to fit all students?

num_classes = num_students // num_per_class
# 14
# but there are some students left!
left_students = num_students % num_per_class
# 12
# these 12 don't have a class. You can decide what to do here.


More conversions, from numbers to strings
number_as_a_string = '3.4'
float(number_as_a_string)
# 3.4
int(number_as_a_string)
# error, you can't do two conversions in one step
int(float(number_as_a_string))
# 3
# You need to nest these conversions
```

## Exercise

```
first  = 1.0
second = '1'
third  = '1.1'

What is the result of...
first + float(second)          # 2.0

float(second) + float(third)    # 2.1

first + int(third)             # error
first + int(float(third))      # 2.0
int(first) + int(float(third))  # 2

2.0 * second          # error, only int can multiply str
2 * second            # '11'



Comments begin with a hash
# ignore this line

Function calls have parentheses (brackets) containing zero or more arguements
print('Hello!')
Can separate multiple arguements with commas, eg 4 arguements
print(first_name, 'is', age, 'years old.')
```

Spyder will show you documentation as you type commands.
Documentation tells you how many and which arguements, and what is returned.

Functions can return something.

```
len('Hello!')
```

returns 6
Can assign this value to a variable

```
word_length = len('Hello!')

int('0')
float(1)
str(1)

print('Hi')
print()
print('all!')
# You get a blank line with print() and no arguements
```

Maximum and minimum functions

```
max(1,2,3)    # 3
min(4,5,6)    # 4
max('a', 'b', 'c')  # 'c'
max('a', 'A')       # 'a' (upper case comes first)
```

Rounding, with optional second arguement for number of decimal places

```
round(3.4)     # 3
round(3.6)     # 4
round(3.5)     # 4
round(3.56, 1) # 3.6
```

Built-in help function

```
help(round)
```

# shows the documentation for the round function
It is possible to show formatted help in other ways in Spyder.
Ctrl + i

Python will try to help you with error messages.
Spyder will complain if you have a syntax error before you even run the code.

```
name = 'Francisco
# SyntaxError
name == 'Francisco'
# SyntaxError
print('Hello!'
# SyntaxError
# EOF means End Of File
age = 2 + aege
# NameError
```

### Exercise

What is happening?

```
word = 'blah '
word = max(min(word * 2 + 'blur', 'aaah '), 'ping')
print(word)
```

Let's break it down.

1. Assign 'blah ' to variable called word
2. Nested functions are evaluated from the inner-most bit outwards.
    1. minimum of (word * 2 + 'blur') and ('aaah ')
    2. minimum of 'blah blah blur' and 'aaah '
    3. 'aaaah ' comes first alphabetically, so is the minimum
3. maximum of 'aaah ' and 'ping'
4. 'ping' comes after 'aaah ' alphabetically so is the maxium
5. print('ping')

### Exercise

```
rich = 'gold'
poor = 'tin'
max(rich, poor)
```

which means

```
max('gold', 'tin')
```

'tin' is alphabetically higher, so max

```
# 'tin'

max(len(rich), len(poor))
```

which means

```
max(len('gold'), len('tin'))
max(4, 3)
# 4
```

Default value of variables is None
None is a special value, it means everything and nothing.

### Continue after lunch break

# Attendance, Day2 after lunch

Gina (host)
Francesco (trainer)
Phil (helper)
Silvia (helper)
Lorraine Linton
Sofiah MacLeod
Iain Scherr
Ivet Gazova
Sarah Knowles
Anna Mazurek
Ken Humphreys
Nicola Raine
Jennifer Ozers
Siddharth Manohar
Monica Mastrantonio
Carol Campbell

## More code from today

Libraries/modules are like building blocks.
There are standard libraries (included with Python) and more.
Bring these in with the keyword import
Keywords are coloured in Spyder

```
import string
```

No error means it was successful.

Use the name of the module then a dot then call something in it.
Could be a value (variable) or a function

```
string.ascii.lowercase
# abcdefghijklmnopqrstuvwxyz
my_name = 'francesco'
string.capwords(my_name)
# 'FRANCESCO'
help(string)
help(string.capwords)
```

Can import a subset, then use the shorter name...

```
from string import ascii_letters
ascii_letters
# abcdefghijklmnopqrstuvwxyz
capwords(my_name)
# 'FRANCESCO'
```

You can create an alias for the module names

```
import string as s
```

Now you can access all the string modules with s then a dot.

```
s.ascii_letters
# abcdefghijklmnopqrstuvwxyz
```

Good practice is to longer version or aliased version. Keep track better. Avoid clashes between names.

Another important library is os for operating system underling features.

```
import os
os.getcwd()
#returns current working directory
help
```

## Exercise

From the Python standard library, which module would help you to use dates?
Index of Python standard library
https://docs.python.org/3/library/

# datetime

```
import datetime
year = 2022
month = 6
day = 8
datetime.date(year, month, day) # produces a new datatype datetime.date
```

To produce this date in ISO format:

```
datetime.date(year, month, day).isoformat()
# '2022-06-08'
```

You can chain functions, eg start from 2nd position

```
datetime.date(year, month, day).isoformat()[2:]
# '22-06-08'
```

If you get an error like this:
NameError: name 'os' is not defined
you will need to include import os

## Activity

fill the blanks

```
import string as s
numbers = ___.digits
print(_____)
```

```
import string as s
numbers = s.digits
```

```
print(numbers)
```

## Activity

Three ways of importing,
A) from string import digits
B) import string
C) import string as s

and three commands
1. print(list(s.digits))
2. print(list(digits))
3. print(string.ascii_uppercase)

Which will work?
A -> 2
B -> 3
C -> 1

To use anything within a library, you need to import it first.

### Lists

Lists are another data type. They can hold multiple values.
We use square brackets. Each element is separated by commas.
Imagine here you have a list of temperature readings from a thermometer.

```
temperatures = [17.3, 17.5, 17.7, 17.5, 17.6]
len(temperatures)  # number of elements ie 5

another_list = ['hello', 'world']
len(another_list)   # number of elements ie 2
```

Lists can be indexed like strings

```
temperatures[0]    # first item
temperatures[1]    # second item
temperatures[0:3]  # items 0, 1, 2
temperatures[0:-1] # all items except last
temperatures[-1:]  # last item only
```

index of lists return an int
slices of lists return a list

reassign values within a list

```
temperatures[0] = 16.5
temperatures    # first item has changed
```

```
temperatures.append(17.9)
# You have added an item to the end of the list
```

To see what methods there are,

```
help(list)
```

You can remove objects from a list:

```
del temperatures[0]
```

The first item has been removed

It is very common to see lists start off empty

```
results = []
len(results)   # 0
```

The program will append objects to the list later.

You can mix types within a list

```
my_list = [1, 'a string', 2, 'A longer string, with weird chars ;D', 3]
```

This is a good thing and a bad thing!
Pay attention.

A minor difference between lists and strings:
You cannot reassign characters within a string.

```
temperatures[0] = 17.9
my_name[0] = 'F'    # Error
```

Strings are immutable.

If you go out of range with an index, you will get an error

```
temperatures[12]   # IndexError: list index out of range
```

## Activity

What would I type here?

```
values =
values.___(1)
values.___(3)
values.___(5)
```

```
values = []      # start with empty list
values.append(1) # append each item
values.append(3)
values.append(5)
# values = [1, 3, 5]
```

What happens here?

```
values[1:3]
# [3, 5]
```

## Continue after break

Length of a slice is end_index minus start_index

Split up a list
```
word = 'tin'
splitted = list(word)
```

Joining up is counter-intuititve
```
''.join(splitted)      # 'tin'
```
Joins up the letters of splitted using the letters of '' which is empty this time.

If you have a different start...
```
'-'.join(splitted)     # 't-i-n'
```

What happens here?
```
list('some string')   # ['s', 'o', 'm', ... ]
```

What happens here?
```
':'.join(['a', 'b'])  # 'a:b'
```

What kind of indexing do I need to take the last letter?
```
word = 'a longer string'
word[-1]
```

Every other letter, add the step (another colon)
```
word[::2]
```
Or start at second, end by 4th, every second letter
```
word[1:4:2]
```

To reverse, use a negative step
```
word[::-1]
```

How do you take all the even characters? Start from second char (index 1)
```
word[1::2]
```

Sorting: there are two ways to write it, they work differently

```
letters = list('gold')
results = sorted(letters)
results = letters.sort()
```
What is the difference?

sorted(letters) will return a sorted copy (new object) -- letters does not change.
letters.sort() will sort in-place -- in other words the letters object changes.

**Dataframe info:**

Very summarised:
https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf
Slightly more explanation
https://www.datacamp.com/cheat-sheet/pandas-cheat-sheet-for-data-science-in-python

Also, you can save files as csv/tsv using the pandas library (if you google these functions you will find lots of information and options).

*import pandas as pd # Import the library*
*df = pd.read_csv('data.csv') # load your data into the dataframe 'df' in this case*

*# do operations on data*

*df.to_csv(index=**False**) # save your data*

# Feedback for day 2

very useful session. Clear and concise and interactive, Really enjoyed it. Good pace for me. Thanks so much :)

# Day 3: Python

## Time schedule Thursday

- **10:00-10:15 Intro**
- **10:15 -11:00 Python (Francesco)**
- **11:00-11:15 Coffee break 1**
- **11:15-12:00 Python (Francesco)**
- **12:00-13:00 Lunch break**
- **13:00-14:00 Python (Francesco)**
- **14:00-14:15 Coffee break 2**
- **14:15-15:00 Python (Francesco)**

## Attendance, Day 3 (write your name):

Gina (host)
Tom Hender (Helper)
Phil Reed (helper)
Francesco Tabaro (trainer)
Lorraine Lint
Ivet Gazova
Nicola Raine
Ken Humphreys
Iain Scherr
Jennifer Ozers
Sofiah MacLeod
Carol Campbell
Anna Mazurek
Sarah Knowles
Siddharth Manohar
Monica Mastrantonio

### Recap from yesterday

Assigning values to variables.
Types include integer, float (decimal), string, list.
We can index lists and strings to pull out specific elements or characters.

```
age = 42
```

```
name = 'Francesco'
person = [name, age]
```

name[1]
Which letter will be printed? 'r' since we count from zero

```
person[0]
```
What is returned? 'Francesco'

We can do slices
```
name[1:4]
```
What is returned?  'ran'
```
person[:1]
```
What is returned? ['Francesco'] Start at beginning, stop before position 1

Check the type of variables
```
type(age) # int
age_float = 42.0
type(age_float) = float
```

Functions with brackets after
```
help(type)
min(...)
max(...)
sorted(...)
round(...)
```

Methods of variables, also with brackets after
```
person.append(age_float)
person.sorted()
```

There is a difference between sorted() and sort()
```
numbers = [12, 100, 50, 75]
sorted_numbers = sorted(numbers) # returns a new list in the sorted order
numbers.sort() # will sort the list in-place
```

Libraries need to be imported
Several ways to do it
```
import os               # entire library
os.getcwd()

from os import getcwd   # specific thing within the library
getcwd()

import string as s      # using an alias
s.ascii_letters
```

## Code from today

'For' loops, to do the same thing over again. You have to indent...

```
numbers = [12, 100, 50, 75, 1234, 1, 17, 90, 15, 63]

for number in numbers:
    • print(number)
```

...prints each number of the list, one per line.
Equivalent to

```
print(numbers[0])
print(numbers[1])
print(numbers[2])
print(numbers[3])
…
```

If you miss out the indentation:

```
for number in numbers:
print(number)
# IndentationError: expected an indented block
```

If you miss out the colon:

```
for number in numbers
    • print(number)
# SyntaxError: invalid syntax
```

The list is 'iterable'. You can do a for loop over any iterable object.

You can have several lines within the for loop.

```
for number in numbers:
    • new_number = numbr + 1
    • print(number, new_number)
```

There is a built-in function range() to generate numbers.

```
for number in range(0, 3):
    • print(number)
for number in range(0, 3):
    • print('Hello!')
```

We can use the 'accumulator pattern' to build/add things up.

```
total = 0
for number in range(10):  # same as for number in range(0, 10)
    • total = total + (number + 1)
print(total)
```

# 55
Takes each number from 0 to 9, adds one (so it is like counting from 1 to 10), then adds it to the previous running total.

Can we count the number of letters in a word using a for loop?

```
word = 'tin'

total = 0
for letter in 'tin':
    • total = total + 1
print (total)
```

# 3

You can use the new variable from the for statement line afterwards (letter)
It will keep its final value.
You can't use it before it is defined.
Maybe use a disposable variable (a name which you won't use outside of the loop by mistake).

## Activity

```
original = "tin"
result = ____
for char in original:
    • result = ____
print(result)
# tin
```

Fill the blanks to make the output print 'tin' without cheating.

```
original = "tin"
result = ""
for char in original:
    • result = result + char
print(result)
# tin
```

## Activity

```
total = 0
for word in ["red", "green", "blue"]:
    • ____ = ____ + len(word)
print(total)
# 12
```

Fill in the blanks to make the total length of all words

```
total = 0
for word in ["red", "green", "blue"]:
```

- `total = total + len(word)`

```
print(total)
# 12
```

## Activity

```
# List of word lengths: ["red", "green", "blue"] => [3, 5, 4]
lengths = ____
for word in ["red", "green", "blue"]:
    lengths.____(____)
print(lengths)
```

Fill in the blanks

```
lengths = []
for word in ["red", "green", "blue"]:
    lengths.append(len(word))
print(lengths)
# [3, 5, 4]
```

## Activity

```
# Concatenate all words: ["red", "green", "blue"] => "redgreenblue"
words = ["red", "green", "blue"]
result = ____
for ____ in ____:
    ____
print(result)
```

Fill in the blanks

```
# Concatenate all words: ["red", "green", "blue"] => "redgreenblue"
words = ["red", "green", "blue"]
result = ""
for word in words:
    result = result + word
print(result)
```

## Back after morning coffee break

You can shorten
result = result + word
as
result += word

## Activity

Reorder these lines, use appropriate indents, to create this output: [1, 3, 5, 10]

```
cumulative += [sums]
for number in data:
cumulative = []
sums += number
print(cumulative)
sums = 0
data = [1,2,2,5]
```

One possible solution:

```
sums = 0
cumulative = []
data = [1,2,2,5]
for number in data:
```

- sums += number
- cumulative += [sums]

```
print(cumulative)
# [1, 3, 5, 10]
```

Notes that these two lines mean the same thing:

```
cumulative += [sums]
cumulative.append(sums)
```

## Activity

```
seasons = ['Spring', 'Summer', 'Fall', 'Winter']
print('My favorite season is ', seasons[4])
```

What error (if any) is there above? How would you fix it?

Index 4 does not exist. Choose a different index.

### Functions

```
def print_greetings():
```

- print("Hello"!)

Defining a function (above) does not run it. You need to call them.

```
print_greetings()
# Hello!
```

You need the indentation, the parentheses and the colon.
Any parameters (arguements) appear in the parentheses.

```
def print_date(year, month, day):
    joined = str(year) + '/' + str(month) + '/' + str(day)
    print(joined)


print_date(2022, 6, 9)
# 2022/6/9
```

The function above prints the result out to the console.
Functions can do operations for us then give back the results.

Corrected code (there is an error in the lesson notes)

```
def average(values):
    total = 0
    for value in values:
        total += value
    return total / len(values)


a = average([1, 3, 4])
print(a)
```

You can return from a function at any time. Usually at the end, or when testing the parameters at the start.
There is a special object None of its own type. It represents nothing.

```
type(None)  # NoneType

def print_greetings():
    print("Hello!")
    return None

# explicit that nothing is returned

def report(pressure):
    print('pressure is', pressure)

report(22.5)
# pressure is 22.5
```

### Activity

```
result = print_date(1871, 3, 19)
print('result of call is:', result)
```

prints

```
1871/3/19
result of call is: None
```

# Attendance, Day3 after lunch

Gina (host)
Lorraine Linton
Jennifer Ozers
Tom Hender (Helper)
Iain Scherr
Ivet Gazova
Ken Humphreys
Anna Mazurek
Phil Reed (helper)
Francesco (instructor)
Sarah Knowles

### Code for the final afternoon

Function definition and call

```
def print_date(year, month, day):
    joined = str(year) + '/' + str(month) + '/' + str(day)
    print(joined)

print_date(1871, 3, 19)
```

We know which arguments are which based on the order.  (You can specify a different order.)

### Activity

```
limit = 100

def clip(value):
    return min(max(0.0, value), limit)

value = -22.5
print(clip(value))
```

What is the value of all variables?

- >>**value is None, limit is None**
- limit = 100
- >>**value is None, limit is 100**
- def clip(value):
    - return min(max(0.0, value), limit)
- >>**value is None, limit is 100**
- value = -22.5
- >> **value is now -22.5, limit is 100**
- print(clip(value))
- >> **now we step into the clip() function...**
    - def clip(value):
        - >> **value is -22.5, limit is 100**
        - return min(max(0.0, value), limit)

# Activity

```
def another_function
  print("Syntax errors are annoying.")
   print("But at least Python tells us about them!")
  print("So they are usually not too hard to fix.")
```

What are the errors and how do you fix them?

```
def another_function():
```
>> add the parentheses and the colon on the first line
>> also fix the indentation on the third line

When you define a variable inside a function, you can only use it within that function.
If you define a variable outside of a function, it is called a global variable. These can be accessed anywhere in the script after they have been define.
This is different to variables defined within a loop.

**Conditionals**

Use the if statement
Control whether a block of code is executed.
It decides this based on a test.
Indent the block of code after the test. (There is a colon too.)

```
mass = 3.54
if mass > 3.0:
    • print(mass, 'is larger than 3.0')
# 3.54 is larger than 3.0

mass = 2.07
if mass > 3.0:
    • print(mass, 'is larger than 3.0')
# no output, the block with the print function is not run
```

Use a list in a for loop with a test

```
masses = [ 3.54, 2.07, 9.22, 1.86, 1.71 ]
for m in masses:
    • if m > 3.0:
        • print(m, 'is larger than 3.0')
# 3.54 is larger than 3.0
# 9.22 is larger than 3.0
```

Can have alternative choice with else statement

```
masses = [ 3.54, 2.07, 9.22, 1.86, 1.71 ]
for m in masses:
    • if m > 3.0:
        • print(m, 'is larger than 3.0')
    • else:
        • print(m, 'is smaller than or equal to 3.0')
# There is a line of output for each element of masses
```

Can have intermediate tests with elif statement (elseif)

```
masses = [ 3.54, 2.07, 9.22, 1.86, 1.71 ]
for m in masses:
    • if m > 9.0:
        • print(m, 'is HUGE and greater than 9.0')
    • elif m > 3.0:
        • print(m, 'is larger than 3.0')
    • else:
        • print(m, 'is smaller than or equal to 3.0')
# prints more 'accurate' output
# 3.54 is larger than 3.0
# 2.07 is smaller than or equal to 3.0
# 9.22 is HUGE and greater than 9.0
# 1.86 is smaller than or equal to 3.0
```

# 1.71 is smaller than or equal to 3.0

After running one block within if, elif, else... jump past the final else block.

Some blocks may never be executed.
The order in which you write tests matters.

Let's say we want to look at the velocity 5 times and adjust

```
velocity = 10.0
for i in range(5):
    print(i, ':', velocity)
    if velocity > 20.0:
        print('velocity is too high! Reducing...')
        velocity = velocity - 5.0
    else:
        print('Moving too slow... Increasing!')
        velocity = velocity + 10.0
print('Final velocity:', velocity)
```

# velocity is 30 at the end

We could have used >= for greater than or equal to.
Or <= for less than or equal to.

We can combine tests ... using and, or, not, parentheses
Here are two vectors of the same length (5):

```
mass     = [  3.54,  2.07,  9.22,  1.86,  1.71 ]
velocity = [ 10.00, 20.00, 30.00, 25.00, 20.00 ]

for i in range(5):
    if mass[i] > 5 and velocity[i] > 20:
        print('Fast and heavy object. Duck!')
    elif mass[i] > 2 and mass[i] <= 5 and velocity[i] <= 20:
        print('Normal traffic')
    elif mass[i] <= 2 and velocity[i] <= 20:
        print('Slow light object. Ignore it!')
    else:
        print('Whoa! Something is up with the data. Check it!')
```

Prints out:

```
Normal traffic
Normal traffic
Fast and heavy object. Duck!
Whoa! Something is up with the data. Check it!
Slow light object. Ignore it!
```

Can use parentheses to group the conditions, like in maths.
1 + 2 / 5  is different to (1 + 2) / 5

```
if (mass[i] > 5 or velocity[i] > 20) and mass[i] <= 10:
```
^^ evaluates the innermost parentheses first then works outwards.

## Side notes: other libraries

For CSV files, there is a standard library csv which is very low level.
```
help(csv)
```
There are non-standard but defacto standards for data analytics
```
numpy
```
scipy
```
pandas        # tabular data
matplotlib    # create charts
seaborn       # create charts
sklearn       # scikit machine learning infrastructure
skimage       # scikit image (computer vision)
nltk          # natural language text analysis, very large packages
```

## Afternoon break

```
pressure = 71.9
if pressure > 50.0:
    pressure = 25.0
elif pressure <= 50.0:
#    pressure = 0.0
print(pressure)
```

What does this print?
25.0

## Activity

```
original = [-1.5, 0.2, 0.4, 0.0, -1.3, 0.4]
result = ____
for value in original:
    if ____:
        result.append(0)
    else:
        ____
print(result)
```

Fill in the blanks so that the output is a new list containing zeroes where the original list's values were negative and ones where the original list's values were positive.

```
# [0, 1, 1, 1, 0, 1]


original = [-1.5, 0.2, 0.4, 0.0, -1.3, 0.4]
result = []
for value in original:
    if value < 0:         # if value is negative
        result.append(0)
    else:
        result.append(1)
print(result)
# [0, 1, 1, 1, 0, 1]
```

## Activity

```
values = [ 1, 2, 3, 10, 50, 42, 21, 12, 123, 96, 100 ]
smallest, largest = None, None
for v in values:
    if ____:
        smallest, largest = v, v
    ____:
        smallest = min(____, v)
        largest = max(____, v)
print(smallest, largest)
```

Modify this program so that it finds the largest and smallest values in the list no matter what the range of values originally is.

We are defining/assigning multiple variables on the same line using the comma

```
smallest, largest = v, v
```

```
values = [ 1, 2, 3, 10, 50, 42, 21, 12, 123, 96, 100 ]
smallest, largest = None, None
for v in values:
    if smallest is None:          # initialise
        smallest, largest = v, v
    else:
        smallest = min(smallest, v)
        largest = max(largest, v)
print(smallest, largest)
```

Cannot compare between None type and int type. smaller and largest start as None but v is always an int.
We want smallest and largest to be int after we get started.
Cannot use starting value of 0 in case the values contain 0 or negative numbers.
In this script, there is no situation when largest is None but smallest is not None.

```
import random
```

```
values = []
for i in range(100):
```

- values.append(random.randit(0,100)

....then copy the above

If you have a string which needs to run over multiple lines, use this format:

```
address = """10 Downing Street,
London,
United Kingdom"""
```

When it prints out, the line breaks may appear like this: \n

```
'10 Downing Street,\nLondon,\nUnited Kingdom'
```

**Style guide PEP8** https://www.python.org/dev/peps/pep-0008
80 characters per line max
keep import statements at the top
and more

Use assertions to make sure of certain things

```
def calc_bulk_density(mass, volume):
    '''Return dry bulk density = powder mass / powder volume.'''
    assert volume > 0
    return mass / volume
```

Doc strings are special comments that go with function definitions (see above) '"
They can run over multiple lines.
First string after the function is define is used, and any others with "'triple single quote"'

```
def average(values):
    '''Return average of values, or None if no values are supplied.'''

    if len(values) == 0:
        return None
    return sum(values) / average(values)

help(average)
```

## Activity

```
"Find maximum edit distance between multiple sequences."
# This finds the maximum distance between all sequences.

def overall_max(sequences):
    '''Determine overall maximum edit distance.'''

    highest = 0
```

```
    for left in sequences:
        for right in sequences:
            '''Avoid checking sequence against itself.'''
            if left != right:
                this = edit_distance(left, right)
                highest = max(highest, this)

    # Report.
    return highest
```

Where is the error?

```
sequences = ['hello', 'world']
overall_max(sequences)
# NameError: name 'edit_distance' is not defined
```

Would need to import it from from a custom module, or type it out above.

```
def edit_distance(left, right):
```

- ...

or

```
import edit_distance
```

# Links to other materials for learning more Python

More similar lessons on Python from the Carpentries (some under development):
Programming with Python, Plotting and Programming with Python
http://swcarpentry.github.io/python-novice-inflammation
http://swcarpentry.github.io/python-novice-gapminder
Video 3 hours: https://youtu.be/JC8c-gE9uM0
Video series with Shell, Python, Git: https://www.youtube.com/playlist?list=PLpX1jXuNTXGoDO5vuHR_pSyxNBr_M233-

Data analysis and Visualization with Python for Social Sciences
https://datacarpentry.org/python-socialsci/

Text analysis in Python (very early stages of development!)
https://carpentries-incubator.github.io/python-text-analysis/

Programming Historian lessons in Python
https://programminghistorian.org/en/lessons/?topic=python

Python website https://www.learnpython.org/

Udemy has a couple of free courses:
this one is a good all rounder, with some emphasis on theory
https://www.udemy.com/pythonforbeginnersintro/
this one is good for absolute beginners
https://www.udemy.com/python-for-absolute-beginners-u/
this one for people who want to dive right into a project
https://www.udemy.com/python-programming-beginners/

SoloLearn
https://www.sololearn.com/Course/Python/

# Post workshop survey

https://carpentries.typeform.com/to/UgVdRQ?slug=2022-06-07-wfd-dc-online

## Feedback