Cheatsheet for coding

We have made you a cheatsheet to help you remember the most important functions from earlier sessions.

General Python syntax

Comments

Comments start with a # and are not executed. You can put a # in front of code so it is now longer see as code to run.

```
# This is a comment and will be ignored by Python
```

Assigning a variable

Assign the number 5 to a new variable called `a`.

```
a = 5
```

= vs ==

One equals sign (=) is used to assign something to a variable.

== is used to test if two things are equal and will return True or False (a boolean!)

```
a = 5
b = 2

# a is not the same as b so this will return False
a == b
```

Printing

If we now print `a`, we should get 5. We can print a using:

```
print(a)
```

You can also print other things like lists or dataframes.

Variable types

Strings

Strings are a type of variable used for text. You have to use quotes (' or ") around the text so Python recognizes it as a string (otherwise it would look for another variable called `string`!).

```
a = 'string'
a = "string"
```

Boolean values

Booleans are a special type of data that can only be `True` or `False`.

```
b = True

# Or resulting from a statement:
c = 3 + 3 == 6 # c will also be equal to True
```

for loops

When you want to perform an action multiple items.

Don't forget the : and the indents necessary for a for loop.

```
for item in item_list:
    print(item)
```

if/else statements

When you want to perform an action only if a certain condition is met.

For example, if you want to check if a number is below 20:

```
if number < 20:
    print("Number is below 20!")
else:
    print("Number is not below 20.)
```

Lists

A special and often used type of variable is a list. It allows you to store multiple values in one variable.

```
example_list = [2, 4, 6, 8, 10]
```

You can select an element from the list using an index. The index of the first element is always 0 in Python.

```
first_item = example_list[0]
```

You can also select multiple items from a list (this is called a 'slice'):

```
# Selecting everything until item 2
items_until2 = example_list[:2]

# Selecting everything FROM item 2 onwards
items_from2 = example_list[2:]

# Selecting items 2, 3 and 5 (the end argument is exclusive)
items_between = example_list[2:6]
```

Functions

Defining a function

To create a function, type "def", then the name of your function, and then two brackets () and : If your function needs any input parameters, you can store that information between the brackets.

For example, let's make a function called `select_first_part` that has two inputs. The first input `list_of_numbers` is a list of numbers and the second `amount` is how many we will select from the list.

We use `return` at the end of the function to return to us any output

```
def select_first_part(list_of_numbers, amount):
    first_part_list = list[:amount]
    return first_part_list
```

Notice how everything inside the function is indented.

A function does not do anything by itself. It is like a recipe: only if you call it and give it input, it will do something.

Calling a function

If we want to use the function we just made, then we will 'call' it.

Here we will call the function with as a `list_of_numbers` example_list that we made earlier and as `amount` 3. This means we will select the first 3 numbers from example_list. We will store the output in `first_3_example_list`.

```
first_3_example_list = select_first_part(example_list, 3)
```

When you give input to a function, it will use the order of the input to determine what is what. In this case, the second input is always the `amount`. You defined this in your function.

You can also explicitly define what is what like this:

```
first_3_example_list = select_first_part(list_of_numbers=example_list, amount=3)
```

Overriding variables

Variables can be overridden, so if you first assign a 5 and then assign it 10, then a will now be 10.

```
a = 5
a = 10
```

Python doesn't know this variable or function

Check if you have run the cell where the function or variable has been defined. Otherwise, check if it was overrriden.

Pandas

Importing the library

Often you need libraries that are created explicitly to do something you need done. This way you don't need to code everything from scratch. You can import a library like this:

```
import pandas as pd
```

where the 'as pd' gives the library the "nickname" `pd`. You can give it any nickname you want but usually there are some conventional nicknames that people tend to use such as `pd` for pandas.

Selecting columns

To select one column, put the column name in square brackets behind the name of the dataframe. To select multiple columns, put the column names in a list instead, which gives you double square brackets.

```
column = df['label']
two_columns = df[['label', 'price']]
```

Selecting rows

You can select rows by their row indices, returning all column values or only a specific subset:

```
# This returns rows 10-20, all columns:
df.loc[10:20]

# This also returns rows 10-20, all columns:
df.loc[10:20, ]

# This returns rows 10-20, only columns label and price:
df.loc[10:20, ['label', 'price']]
```

We can also select rows by a certain condition by passing the condition to `.loc` rather than a range of row indices: `df.loc[condition, ]` will select all the items where the condition returns True (and all columns).

E.g., to select all rows that have value 1 in column label, our condition becomes `df['label'] == 1`, and to select save those rows to a new dataframe, we write:

```
# Keeping all the columns
df2 = df.loc[df['label'] == 1]

# Or selecting a subset of columns as well as rows
df2 = df.loc[df['label'] == 1, ['label', 'price']]
```

Loading a csv file with pandas

Make sure the file is in the same folder as the notebook you are running, or specify the correct path to the file:

```
df = pandas.read_csv(file)
```

Inspecting the first or last part of a dataframe

To get the first few rows of a dataframe, use `df.head()`. The default number of rows is 5, if you want a specific number of items like 10 you can use `df.head(10)`.

To get the last few rows of a dataframe, use `df.tail()`. The default number of rows is 5, if you want a specific number of items like 10 you can use `df.tail(10)`.

See the shape of a dataframe

To see the shape (length of rows and columns), use `df.shape`. This returns a list with the number of rows and columns.