# 1. Numbers

1. Open the python shell by typing `python`
2. Try using the shell as a calculator
3. Try `10/3`
4. Try `10/3.0`
5. Whats the type of `10`? Of `10.0`?

# 2. Booleans

1. Try the >, =, < operators
2. Can you compare a string to a float?
3. Type `a = 3`. Now type `a`. What's the type of `a`?
4. Type `b = 3 > 2`. Now type `b`. What's the type of `b`?
5. What's the difference between `a = 3` and `a == 3`?
6. Try `c == 10`. What happens? Why?
7. Type `c = 5`, then again `c == 10`. What happens now?

# 3. Strings I

1. Whats the difference between `"a"` and `a`?
2. Try `a = "Python"`, then `b = "Course"`, then `c = a + b`
3. Whats the result of `"on" in c`? `"Course1" in c`?
4. Try slicing on the previous variable c. Whats c[-1]? How do you get "Py"? "course"? "Ptocus"?

# 4. Strings II

1. Try the string methods. What happends if you forget the brackets?
2. BONUS: `d = "python"`.
   How do you change d to "Python" in one command using slicing and string methods?
   Remember that you can add strings with "+" and generally combine almost all operations with each other

## 5. Lists

Remember the difference between a variable and a string!

1. Create a list of fruits in the python shell e.g `fruits = ["apple", "apple", "orange", "banana"]`
2. Try the slicing operations you learned for strings.
3. Try the "x in y" comparison to test if a fruit is in your list or not.
4. How do you include "pineapple" in your list?
5. Try to join your list into a long string using e.g. `"-".join(fruits)`
6. Count the amount of apples in your list
7. BONUS: Create a second list of veggies. How do you combine the lists into one?

## 6. Dictionaries

Remember the different types of brackets!

1. Create a dict of students and their grades `{"Tom":"A", "John":"C", "Martin":"F-"}`
2. Which are the keys and which are the values? What's the difference?
3. Retrieve the grade for Martin using dict[key].
4. Add new student "Andreas" who has the grade "A". Print the dict.
5. Add another student called "John" who has the grade "A". Print the dict. What happened?
6. BONUS: Find out how many students have the grade "A".

## 7. Simple scripting

1. Open your text editor of choice.
2. Write `print "Hello World!"`
3. Save as hello_world.py
4. In the terminal: Make sure you are in the same folder as the file.
5. Type `python hello_world.py`. Congratulations!

## 8. For loops

Remember the indentation!

1. Go back to you Hello_World.py script. Save a copy as hello_world_for_loop.py.
2. Create a list of people. Print each name using a for-loop.
3. BONUS: Have the script individually say hello e.g. "Hello Tom!", "Hello John!" etc. using string substitution.

## 9. File input and output

1. Write a simple replacement for "grep". Hints: open a file, iterate over the rows, check for you search term, print row if found.
2. BONUS: Have the script print the row number of each output row together with the row.
   Hint: a numerical variable can be increased by e.g. `a += 1` !

## 10. Modules

1. Extend your grep script to accept the search term as a commandline argument.
2. BONUS: Have the script find out if the user gave an argument and use a default search term if he didn't. Hint: the arguments are parsed into a list, you can find out the length of a list with len(list).

## 11. Functions

1. Write a function that takes a name and prints a nice greeting. Test it on a list of names.
2. BONUS: Extend your grep script to include a function that tests if the search term is in a row.

# 12. Real world example

Many basic bioinformatics tasks involve reading information from one or more files, performing some kind of filtering or comparison and writing the results into another file. In this example, imagine you have a huge list of variants and a colleague asks you to prepare a smaller list of good variants in her favorite gene.

We will try such a task using our "quasar.tsv" file from the Bash exercise. Build your code slowly by testing if each row you write does what you think it does (use a `print` statement that you delete later). Try to get each task right before moving on to the next.

1. Write a script to read the quasar file and print all rows.
2. Only show rows in TP53. Hint: you can test if a string contains another string with `X in Y`.
3. Only show rows with a quality above 40. Hint: you need to transform the QC column into a variable of the right type.
4. Use your bash skills to cross-check the number of rows found by Python and by a bash pipe. Do they match?
5. Output your results into a new file using the write() method. You could of course use a ">" on the shell, but don't be lazy!
6. BONUS 1: Parse the name of a target gene using the sys.argv[] method, e.g. TP53. Then find the rows for this gene and write them into an output with a custom name of the format "variants_in_GENENAME.tsv" e.g. "variants_in_TP53.tsv".
7. BONUS 2: Only show rows with genes present in the file "target_genes.txt". Hint: you need to first parse these genes into the proper Python data structure. Then you can use it to test if a row contains a target gene or not.
8. BONUS 3: Combine Bonus 1 and 2 to automatically create a matching output file for each of the target genes.