Lecture 1 - Python basics

University of California, Berkeley - Spring 2022

My first python script

In [1]:
```
"Hello ATGC!"
```

Out[1]:
```
'Hello ATGC!'
```

What can we do with python?

Use python as a calculator

We can peform simple calculations

In [2]:
```
4
```

Out[2]:
```
4
```

In [3]:
```
4+5
```

Out[3]:
```
9
```

In [4]:
```
(4+2-11)*3
```

Out[4]:
```
-15
```

In [5]:
```
from math import exp
exp(-9)
```

Out[5]:
```
0.00012340980408667956
```

In [6]:
```
from math import sin,pi
sin(pi/3)
```

Out[6]:
```
0.8660254037844386
```

Print stuff to the screen

We can just print text

In [7]:
```
print("Welcome to python for life-science course!")
```

```
Welcome to python for life-science course!
```

Or we can print text along with some calculation

In [8]:

```
print("The product of 7 and 8 is",7*8)
```

```
The product of 7 and 8 is 56
```

Variables

We can store *values* in the computer's memory, instead of just calculating/printing them. Values are stored within *variables*.
A variable always has:

- A name
- A value
- A defined type (number,text,etc.)

To insert a *value* into a *variable*, we use *asignments*, specifically using the '=' sign.

In [10]:
```
a = 5
```

Once a variable has been declared, we can use it to get its value.

In [11]:
```
print(a)
```

```
5
```

In [12]:
```
a + 7
```

Out[12]:
```
12
```

We can assign new variables

In [13]:
```
b = a * 2
```

In [14]:
```
a + b
```

Out[14]:
```
15
```

We can assign a new value to an existing variable, overwriting the previous value.

In [15]:
```
print("a is",a)
a = 8
print("and now a is",a)
```

```
a is 5
and now a is 8
```

What happens to b???

In [16]:
```
print(b)
```

```
10
```

We can determine a variable's type using the type() command. There are integers (int type)

In [17]:
```
type(a)
```

Out[17]:
```
int
```

Strings (text) - we'll talk more about strings next time.

In [18]:
```
seq = 'ATGCGTATAGCAGATACAGt'
type(seq)
```

Out[18]:
```
str
```

floating point (real numbers)

In [19]:
```
pi = 3.14159265359
type(pi)
```

Out[19]:
```
float
```

and another thing, called boolean variables, which get either *True* or *False*. We'll come back to these soon.

In [20]:
```
booly = True
type(booly)
```

Out[20]:
```
bool
```

Some notes about variable names:

- Make them meaningful! Instead of using *x*, *y*, *a*, *b* etc, choose names that will resemble the meaning of the variable, such as _sequence*length*, _sum_of*numbers* and so on.
- You can choose any name, but you can't include spaces, special characters and words that have special meaning in python (for example *print*).
- If you need to give long variable names, the convention is either to use underscores, or to start each word with a capitall letter - first_sequence_length or firstSequenceLength. We will use underscores throughout the course.

Comments

We can add explanatory text to our code to make it more readable. We do that by simply adding a '#' in the beginning of a comment.

In [21]:
```
print("This will be printed")
# print("This will not be printed")
print("Another example") # of a comment
```

```
This will be printed
Another example
```

Operators

Operators allow us to perform basic actions on variables

Arithmetic operators

Used on numbers (integers and floating point). We've already seen some.

In [22]:
```
num1 = 8
```

```
num2 = 5
```

In [23]:
```
num1 + num2
```

Out[23]:
```
13
```

In [24]:
```
num1 - num2
```

Out[24]:
```
3
```

In [25]:
```
num1 * num2
```

Out[25]:
```
40
```

In [26]:
```
num1 / num2
```

Out[26]:
```
1.6
```

In [27]:
```
num1 // num2
```

Out[27]:
```
1
```

In [28]:
```
num1 % num2
```

Out[28]:
```
3
```

In [29]:
```
num1 ** num2
```

Out[29]:
```
32768
```

Comparisson operators

These operators are used to compare numbers and strings. They always return boolean values, i.e. True or False.

This is pretty straightforward for integers:

In [30]:
```
num1 == num2    # Note: '==', not '='
```

Out[30]:
```
False
```

In [31]:
```
num1 == 8
```

Out[31]:
```
True
```

In [32]:
```
num1 > num2
```

Out[32]:
```
True
```

In [33]:
```
num2 > num1
```

Out[33]:
```
False
```

In [34]:
```
num1 != num2
```

Out[34]:
```
True
```

In [35]:
```
num2 < 5
```

Out[35]:
```
False
```

In [36]:
```
num2 <= 5
```

Out[36]:
```
True
```

For strings, '<' and '>' operators are based on alphabetical order.

In [37]:
```
plant = 'Arabidopsis thaliana'
mammal = 'Mus musculus'
plant == mammal
```

Out[37]:
```
False
```

In [38]:
```
plant > mammal
```

Out[38]:
```
False
```

In [39]:
```
plant <= mammal
```

Out[39]:
```
True
```

Logical operators

These operators work on *booleans* rather than integers or strings. Logical operators always return booleans.

There are three logical operators:

In [40]:
```
(num1 > num2) and (num1 != num2)
```

Out[40]:
```
True
```

In [41]:
```
(num1 != num2) and (num1 < num1)
```

Out[41]:
```
False
```

In [42]:
```
(num1 != num2) or (num1 < num1)
```

Out[42]:
```
True
```

In [43]:
```
num1 == 3 or num2 == 10 or num2 > 7
```

Out[43]:
```
False
```

In [44]:
```
not (num1 > num2)
```

Out[44]:
```
False
```

In [45]:
```
not (num1 < num2)
```

Out[45]:
```
True
```

In [46]:
```
boolean = (num1 > num2)
type(boolean)
```

Out[46]:
```
bool
```

In [47]:
```
(boolean) and num2 == 5
```

Out[47]:
```
True
```

We can also think of logical operators as 2X2 matrices, or alternatively - Venn diagrams.

Branching - IF statements

So far we've seen (small) programs that just start running, and finish when all commands are performed. But sometimes we want to perform certain commands only *if* a condition is met. For this we use *if statements*:

In [48]:
```
if num1 > num2:
    print('Yes')
```

```
Yes
```

In [49]:
```
if num1 < num2:
    print('Yes')
```

Notice the colon and the indented block. The syntax is always:
if *condition*:
    indented commands

Only commands within the indented block are conditional. Other commands will be executed, no matter if the condition is met or not.

In [50]:
```
if num1 > num2:
    print('Yes')
    print('Another operation will follow')
    num1 = 10
print(num1)
```

```
Yes
Another operation will follow
```

```
10
```

Note: the condition expression always returns a boolean, and the indented commands only occur if the boolean has a True value. Therefore, we can use logical operators to create more complex conditions.

In [51]:
```
x = 15
y = 8
if (x > 10 and y < 10) or x * y == 56:
    print('Yes')
```

```
Yes
```

In [52]:
```
x = 9
if (x > 10 and y < 10) or x * y == 56:
    print('Yes')
```

In [53]:
```
x = 7
if (x > 10 and y < 10) or x * y == 56:
    print('Yes')
```

```
Yes
```

Let's write a program that checks if a number is devisible by 17. Remember the Modulus operator...?

In [54]:
```
x = 442
if x % 17 == 0:
    print('Number is devisible by 17!')
print('End of program.')
```

```
Number is devisible by 17!
End of program.
```

We can add *else* statements to perform commands in case the condition is **not** met, or in other words, if the boolean is False.

In [55]:
```
x = 586
if x % 17 == 0:
    print('Number is devisible by 17!')
else:
    print('Number is not devisible by 17!')
print('End of program.')
```

```
Number is not devisible by 17!
End of program.
```

Things get even more interesting when using *elif* statements, where multiple conditions are tested one by one. Once a condition is met, the corresponding indented commands are performed. If none of the conditions is True, the *else* block (if exists) is executed.

In [56]:
```
x = 586
if x % 17 == 0:
    print('Number is devisible by 17!')
elif x % 2 == 0:
    print('Number is not devisible by 17, but is even!')
else:
    print('Number is not devisible by 17, and is odd!')
print('End of program.')
```

```
Number is not devisible by 17, but is even!
End of program.
```

While loops - continuos If

One thing computers are very good at (and most humans not) is doing repetitive jobs.
We use *while* loops to do something again and again, as long as a condition is met.

The syntax is very similar to that of **if** statements.

In [57]:
```
from random import randint
random_num = randint(1,100)
while random_num <= 90:    # condition
    print(random_num)      # indented block
    random_num = randint(1,100)
print ('Found a number greater than 90!', random_num)
```

```
57
Found a number greater than 90! 91
```

When using a while loop, always make sure that you change the value of the variable tested in the condition. Otherwise, the condition will always be `True` and you will find yourself in an infinite loop... For example, here we change the value by obtaining a new random number.

Now let's count how many times it takes to get a random number greater than 90. We'll use a counter variable.

In [58]:
```
from random import randint
counter = 1
random_num = randint(1,100)
while random_num <= 90:    # condition
    print(random_num)      # indented block
    random_num = randint(1,100)
    counter = counter + 1    # what's happening here?
print ('Found a number greater than 90!', random_num, '. It took',counter,'tries.')
```

```
47
38
31
2
41
53
48
26
74
63
Found a number greater than 90! 91 . It took 11 tries.
```

In [59]:
```
m = 555 # integer to apply the conjecture on

n = m
while n != 1:
    print(n, end=", ")
    # if n is even
    if n % 2 == 0:
        n = n // 2
    # if n is odd
    else:
        n = 3 * n + 1
print(1) # 1 was not printed
print(m, "is OK")
```

```
555, 1666, 833, 2500, 1250, 625, 1876, 938, 469, 1408, 704, 352, 176, 88, 44, 22, 11, 34,
17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
```

```
555 is OK
```

Congrats!

The notebook is available at https://github.com/Naghipourfar/molecular-biomechanics/genomics/1-Basics.ipynb