# Recitation 02
# -
# Values, Variables, User Input

## Exercise 1 - All work and no play

Take the sentence: `All work and no play makes Jack a dull boy`. Store each word in a separate variable, then print out the sentence on one line using `print`.

## Exercise 2 - Order of operations

Add parenthesis to the expression `6 * 1 - 2` to change its value from `4` to `-6`.

## Exercise 3 - Adding a comment

Place a comment before a line of code that previously worked, and record what happens when you rerun the program.

## Exercise 4 - NameError

Start the Python interpreter and enter `sam + 4` at the prompt. This will give you an error:

```
NameError: name 'sam' is not defined
```

Assign a value to `sam` so that `sam + 4` evaluates to `10`.

## Exercise 5 - Introducing doctest

Make a file named `doctestintro.py` that contains the following:

```
"""
  >>> n
  42
"""
# Place your solution code on the line after this one...


if __name__== '__main__':
    import doctest
    doctest.testmod()
```

Save the file and run it. When you run this program, it will give you an error that looks something like this:

```
**********************************************************************
File "doctestintro.py", line 2, in __main__
Failed example:
    n
Exception raised:
    Traceback (most recent call last):
      File "/lib/doctest.py", line 1253, in __run
        compileflags, 1), test.globs)
      File "<doctest __main__[0]>", line 1, in <module>
        n
    NameError:  name 'n' is not defined
**********************************************************************
1 items had failures:
   1 of 1 in __main__
***Test Failed*** 1 failures.
```

You will become familiar with these *tracebacks* as you work with Python. The most important thing here is the last line, which tells you that the name `n` is not defined.

To define `n`, assign it a value by adding the following on the line immediately following the `# Place your solution code on the line after this one...` line in the program.

```
n = 11
```

Run the program again, and confirm that you now get an error like this:

```
**********************************************************************
File "doctestintro.py", line 2, in __main__
Failed example:
    n
Expected:
    42
Got:
    11
**********************************************************************
1 items had failures:
   1 of 1 in __main__
***Test Failed*** 1 failures.
```

`n` is now defined, but our *doctest* is expecting its value to be: `42`, not `11`. Fix this by changing the assignment to `n` to be

```
n = 42
```

When you run the program again, you should not get an error. In fact, you won't see any output at all. If you are running your program from a command prompt, you can add `-v` to the end of the command to see *verbose* output:

```
$ python3 doctestintro.py -v
Trying:
    n
Expecting:
    42
ok
1 items passed all tests:
   1 tests in __main__
1 tests in 1 items.
   1 passed and 0 failed.
Test passed.
```

**Note**

This exercise and the exercises that follow all make use of Python's built-in automatic testing facility call *doctest*. For each exercise, you will create a file that contains the following at the bottom:

```
if __name__== '__main__':
    import doctest
    doctest.testmod()
```

The "tests" will be written at the top of the file in triple quoted strings (called *docstrings*), and will look just like interactions with the Python shell. Your task will be to make the tests pass (run without failing or returning any error).

To run the tests on a unix system, type the command to launch Python 3 (`python3` on most current systems) followed by the file name of your program:

```
$ python3 doctestintro.py
```

It is also possible to run your *doctest* from the Python prompt:

```
>>> import doctest
>>> import doctestintro
>>> doctest.testmod(doctestintro)
```

## Exercise 6 - doctest exercise 2

Make a file named `doctestex2.py` that contains the following:

```
"""
  >>> f
  3.5
"""
# Place your solution code on the line after this one...


if __name__== '__main__':
     import doctest
     doctest.testmod()
```

Make the test pass using what you learned in the previous exercise.

## Exercise 7 - doctest exercise 3

Create a file named `doctestex3.py` that contains the following:

```
"""
  >>> x + y
  42
  >>> type(message)
  <class 'str'>
  >>> len(message)
  42
"""
# Place your solution code on the line after this one...


if __name__== '__main__':
     import doctest
     doctest.testmod()
```

This time there are 3 tests instead of one. You will need more than one assignment statement to make these tests pass.

**Note**

From here on we will just give you the *docstrings* for each exercise. As in the previous exercises, you will need to create a file for each one and put the three lines of Python at the bottom of the file to make the tests run.

## Exercise 8 - Sum and difference

```
"""
  >>> n + m
  15
  >>> n - m
  5
"""
```

## Exercise 9 - Sum and difference 2

```
"""
  >>> n + m
  55
  >>> n - m
  -3
"""
```

## Exercise 10 - Types of things

You will need to create variables of the appropriate type to make these tests pass.

```
"""
  >>> type(thing1)
  <class 'float'>
  >>> type(thing2)
  <class 'int'>
  >>> type(thing3)
  <class 'str'>
"""
```

## Exercise 11 - Types of things 2

Create variables of the appropriate type to make these tests pass.

```
"""
  >>> type(this)
  <class 'str'>
  >>> type(that)
  <class 'int'>
  >>> type(something)
  <class 'float'>
"""
```

## Exercise 12 - Exponentiation

Remember that `**` is the exponentiation operator for this one.

```
"""
  >>> x ** n
  100
"""
```

## Exercise 13 - Modulus

Review the modulus operation before trying this one.

```
"""
  >>> n % 5
  2
"""
```

## Exercise 14 - Three numbers with two restrictions

There are limitless solutions to this. Just find one that makes the tests pass.

```
"""
  >>> a + b + c
  50
  >>> a - c
  10
"""
```

## Exercise 15 - Four numbers with three restrictions

There are limitless solutions to this. Just find one that makes the tests pass.

```
"""
  >>> a + b + c + d
  50
  >>> a - c
  10
  >>> a + b
  20
"""
```

## Exercise 16 - Joining strings

Review string concatenation before trying this one.

```
"""
  >>> s1 + s2 + s3
  'Three strings were concatenated to make this string.'
"""
```