1. Hello, Python

1. Variable assignment : Here we create a variable called spam_amount and assign it the value of 0 using =, which is called the assignment operator.
2. Function calls : print is a Python function that displays the value passed to it on the screen. We call functions by putting parentheses after their name, and putting the input to the function in those parentheses.
3. Comment : In python, comments begin with # symbol.
4. Reassignment : Reassigning the value of an existing variable looks just the same as creating variable - it still uses the = assignment operator.

2. Functions and Getting help

1. Getting help :
   You saw the as function in the previous tutorial, but what if you've forgotten what it does? The help() function is possibly the most importatn Python function you can learn. If you can remember how to use help(), you hold the key to understanding other functions. help() display two things

   - the header of that function round(number, ndigits = None). In this case, this tells us that round() takes an argument we can describe as number. Additionally, we can optionally give a separate argument which could be described as ndigits.
   - A brief English description of what the function does.
2. Defining functions

   - Builtin functions are great, but we can only get so far with them before we need to start defining our own functions. Functions start with a header introduced by the def keyword. THe indented block of code following the : is run when the function is called. return is another keyword uniquely associated with functions. When Python encounters a return statement, it exits the function immediately, and passes the value on the right hand side to the calling context.
   - When we use help() function to personal function it can't tell us something. Python isn't smart enough to read my code and turn it into a nice English description. However, when i write a function, I can provide a description in what's called the docstring.

In [1]:
```
# Docstrings

def least_difference(a, b, c):
    """Return the smallest difference between any two numbers
    among a, b and c.

    >>> least_difference(1, 5, -5)
    4
    """
    diff1 = abs(a - b)
    diff2 = abs(b - c)
    diff3 = abs(a - c)
    return min(diff1, diff2, diff3)

# Functions Applied to functions

def least_difference(a, b, c):
    """Return the smallest difference between any two numbers
    among a, b and c.

    >>> least_difference(1, 5, -5)
    4
    """
    diff1 = abs(a - b)
    diff2 = abs(b - c)
    diff3 = abs(a - c)
    return min(diff1, diff2, diff3)
```

In [2]:
```
help(least_difference)
```

```
Help on function least_difference in module __main__:

least_difference(a, b, c)
    Return the smallest difference between any two numbers
    among a, b and c.

    >>> least_difference(1, 5, -5)
    4
```

3. Booleans and Conditionals

1. Booleans : Python has a type of variable called bool. It has two possible values : True and False. Rather than putting True or False directly in our code, we usually get boolean values from boolean operators. These are operators that answer yes/no questions.

1. Comparision Operations 1) a == b : a equal to b 2) a < b : a less than b 3) a <= b : a less than or equal to b 4) a != b : a not equal to b 5) a > b : a greater than b 6) a >= b : a greater than or equal to b

1. Cobining Boolean Values : You can combine values using the standard concepts of "and", "or", 'Not. In fact, the words to do this are : and, or, and not.

1. Conditionals : Booleans are most useful when combined with conditional statements, using the keywords if, elif, and else. Conditional statements, ofttten referred to as if-then statements, let you control what pieces of code are run based on the value of some Boolean condition. The if and else keywords are often used in other languages; its more unique keyword is elif, a contraction of "else if". In these conditional clauses, elif and else blocks are optional; additionally, you can include as many elif statements as you would like.

1. Boolean conversion : We've seen int(), which turns things into ints, and float(), which turns things into float, so you might not be surprised to hear tha Python has a bool() function which turns things into bools.

4. List

1. Lists : lists in python represent ordered sequences of valeus. A list can contain a mix of different types of variables

1. Indexing : We can access individual list element with square brackets : planets[0], elements at the end of the list can be accessed with negative numbers, starting from -1.

1. Update lists : List are "mutable", meaing they can be modified "in place". One way to modify a list is to assign to an index or slice expressions.

1. List functions : Python has several useful functions for woring with lists.
   - len : gives the length of a list
   - sorted : returns a sorted version of a list
   - sum : does what you might expect

1. List methods :
   - list.append : modifies a list by adding an item to the end
   - list.pop : removes and returns the last element of a list
   - list.index : serarching by indexing

1. Tuples : Tuples are almost exactly the same as list but cannot be modified

5. Loops and List Comprehensions

1. Loops : Loops are a way to repeatedly execute some code. The for loop specifices the variable name to use, the set of value to loop over. You use the word "in" to link them together.

1. range() : range is a function that returns a sequence of numbers. It turns out to be very useful for writing loops.

1. while loops : The other type of loop in Python is a while loop, which iterates until some condition is met. The argument of the while loop is evaluated as a boolean statement, and the loop is executed until the statement evaluates to False.

1. List comprehensions : List comprehensions are one of Python's most beloved and unique features. The easiest way to understand them is probably to just look at a few examples :

In [3]:
```
nums = [3, 3, -1, -3, 0, 3, 2, 7, 8]

res1 = [i for i in nums if i != 3]
res2 = [num < 0 for num in nums]

print(res1, res2, sep = '\n')
```

```
[-1, -3, 0, 2, 7, 8]
[False, False, True, True, False, False, False, False, False]
```

6. Strings and Dictionaries

1. Strings :
   One place where the Python language really shines is in the manipulation of strings. We've already seen plenty of strings in examples during the previous lessons, but just to recap, strings in Python can be defined using either single or double quotations. Double quotes are convenient if your string contains a single quote chacater. The last sequence, \n, represents the newline character. It causese Python to start a new line. In addition, Python's triple quote syntax for strings let us include newlines iterally.

1. Print() : The print() function automatically adds a newline character unless we specify a value for the keyword argument end other tan the default value of '\n'.

1. Strings are sequence : Strings can be thought of as sequence of characters. Almost everything we've seen that we can do to a list, we can also do to a string.

1. String method()
   - upper() : make string's character as up
   - lower() : make string's character as low
   - index() : Searching for the first index of a substring
   - split() : str.split() turns a string into a list of smaller strings, breaking on whitespace by default. This is super useful for taking us from one big string to a list of words.
   - join() : takes us in the other direction, sweing a list of strings up into one long string, using the string it was called on as a separator.

1. Dictionaries : Dictionaries are a built-in Python data structure for mapping keys to values.

In [4]:
```
numbers = {'one' : 1, 'two' : 2, 'three' : 3}
numbers['one']
```

Out[4]:
```
1
```

In this case 'one', 'two', and 'three are the keys, and 1, 2, and 3 are their corresponding values. Values are accessed via square bracket syntax similar to indexing into lists and strings.
Python has dictionary comprehensions with a syntax similar to the list comprehensions we saw in list comprehensions

In [5]:
```
planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
planet_to_initial = {planet: planet[0] for planet in planets}
planet_to_initial
```

Out[5]:
```
{'Mercury': 'M',
 'Venus': 'V',
 'Earth': 'E',
 'Mars': 'M',
 'Jupiter': 'J',
 'Saturn': 'S',
 'Uranus': 'U',
 'Neptune': 'N'}
```

1. Dictionary method() :
   - dict.keys() : call all keys
   - dict.values() : call all values
   - dict.items() : let us iterate over the keys and values of a dictionary simultaneously.

7. Working with External Libraries

1. Imports :
   So far we've talked about types and functions which are built- in to the language. But one of the best things about Python if you're data scientist is the vast number of high-quality custom libraries that have been written for it. Some of these libraries are in the 'standard library', meaning you can find them anywhere you run PYthon. Other libraries can be easily added, even if they aren't alwalys shipped with Python.

In [1]:
```
import math
print(dir(math))
```

```
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'ata
n', 'atan2', 'atanh', 'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'expm
1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'is
inf', 'isnan', 'isqrt', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'perm', 'pi', 'pow', 'pro
d', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

If we know we'll be using functions in math frquently we can import it under a shorter alias to save some typing.

In [2]:
```
import math as mt
```

1. Submodules :
   We've seen that modules contain variables which can refer to functions or values. Something to be aware of is that they can also have variables referring to other modules.

In [4]:
```
import numpy as np
print(dir(np.random))
```

```
['BitGenerator', 'Generator', 'MT19937', 'PCG64', 'Philox', 'RandomState', 'SFC64', 'SeedSequence', '__RandomState_ct
or', '__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__path
__', '__spec__', '_bounded_integers', '_common', '_generator', '_mt19937', '_pcg64', '_philox', '_pickle', '_sfc64',
'beta', 'binomial', 'bit_generator', 'bytes', 'chisquare', 'choice', 'default_rng', 'dirichlet', 'exponential', 'f',
'gamma', 'geometric', 'get_state', 'gumbel', 'hypergeometric', 'laplace', 'logistic', 'lognormal', 'logseries', 'mtra
nd', 'multinomial', 'multivariate_normal', 'negative_binomial', 'noncentral_chisquare', 'noncentral_f', 'normal', 'pa
reto', 'permutation', 'poisson', 'power', 'rand', 'randint', 'randn', 'random', 'random_integers', 'random_sample',
'ranf', 'rayleigh', 'sample', 'seed', 'set_state', 'shuffle', 'standard_cauchy', 'standard_exponential', 'standard_ga
mma', 'standard_normal', 'standard_t', 'test', 'triangular', 'uniform', 'vonmises', 'wald', 'weibull', 'zipf']
```

1. Three tools for understanding strange objects :
   - type() : what is this things?
   - dir() : what can i do with it?
   - help() : tell me more