Basics of Programming in Python

Data types

Definition:

- All programs are composed of two items: Data and Operations on that Data. Because, at their heart, computers are simple devices, they can only represent very simple pieces of information. All complex information must be built up from these basic **Data Types**. The data types can roughly be described as: numbers, booleans, characters, Strings, arrays.
- In many languages, before you declare a variable, you have to declare the type of the variable. However, in Python, you do not need to do that. See the examples below.

Example:

- Numbers:
  - students = 12 % an integer
  - average_test_score = 90.5 % a double
- Booleans:
  - this_class_is_the_best = True
  - henry_is_boring = False
- Characters:
  - 'a', 'c', '?'
- Strings: "Candice", "KIS"
- Arrays:
  - student_names = ["Candice", "Henry", "Alice"]
  - student_grades = [100, 90, 35.5, 87]

Data structures: https://docs.python.org/2/tutorial/datastructures.html

Lists: As we want to solve more complex problems in Python, we need more powerful variables. Up to now we have been using simple variables to store numbers or strings where we have a single value in a variable. Starting with lists we will store many values in a single variable using an indexing scheme to store, organize, and retrieve different values from within a single variable. We call these multi-valued variables "collections" or "data structures".

Dictionary: The Python dictionary is one of its most powerful data structures. Instead of representing values in a linear list, dictionaries store data as key / value pairs. Using key / value pairs gives us a simple in-memory "database" in a single Python variable.

Tuples: Tuples are our third and final basic Python data structure. Tuples are a simple version of lists. We often use tuples in conjunction with dictionaries to accomplish multi-step tasks like sorting or looping through all of the data in a dictionary.

Example:

List: student_names = [“Candice”, “Henry”, “Alice”]
Dictionary: gradebook = {“Candice”: 50, “Henry”: 100}
Tuple: candice_grade = (“Candice”, 50)

Functions
You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

- Function blocks begin with the keyword def followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

```
def functionname( parameters ):
     "function_docstring"
     function_suite
     return [expression]
```

Example:

```
def printme( str ):
     "This prints a passed string into this function"
     print str
     return
```

Control flow: https://en.wikibooks.org/wiki/Python_Programming/Control_Flow
Definition
In the programs we have seen till now, there has always been a series of statements faithfully executed by Python in exact top-down order. What if you wanted to change the flow of how it works? For example, you want the program to take some decisions and do different things depending on different situations, such as printing 'Good Morning' or 'Good Evening' depending on the time of the day?

As you might have guessed, this is achieved using control flow statements. There are three control flow statements in Python - `if`, `for` and `while`.

- The `if` statement is used to check a condition: *if* the condition is true, we run a block of statements (called the *if-block*), *else* we process another block of statements (called the *else-block*). The *else* clause is optional.
- The `while` statement allows you to repeatedly execute a block of statements as long as a condition is true. A `while` statement is an example of what is called a *looping* statement.
- The `for..in` statement is another looping statement which *iterates* over a sequence of objects i.e. go through each item in a sequence.

Example

- `if` statement

```
if 1+1 == 2:
      return True
else:
      return False
```

- `while` statement

```
x = 0
while x != 10:
      x += 1
```

- `for..in` statement (`for` loop)

```
students = [“Andrew”, “Jason”, “Rachel”, “Jacob”, “Joshua”, “Yoo Bin”, “Lina”]
for student in students:
      print “hello”+student
```