# INTRODUCTORY PYTHON

2023

Student's Society of McGill University

## 0. Table of Contents



Funding acknowledgement: This open textbook compilation was sponsored by the Student Society of McGill University (SSMU) as a part of the SSMU Library Improvement Fund.

# 1. Introduction

### Binary

The number system we use in everyday life is called the decimal number system and it is base 10 which contains the ten standard digits (0 to 9). The computer uses the binary number system, and it is base 2 because it is built using transistors which can only be charged and discharged (only 2 states), hence, 1s and 0s to represent charged and uncharged. Just like how a number in base 10 can be represented using powers of 10, a number in base 2 can be represented by using powers of 2. We denote the different number systems using the subscript 2 or 10 depending on the base of the number system.

To convert from decimal to binary we use the “divide by 2” algorithm, where we take the decimal number, divide by 2, and record the remainder (0 or 1). Then, divide the result by 2 again, record the remainder, and so on. The binary number is the reversed sequence of the remainders. In the following example, 109 is converted from decimal into binary, resulting in the binary number $1101101_2$.

| | result | remainder |
|---|---|---|
| divide 109 by two | 54 | 1 |
| divide 54 by two | 27 | 0 |
| divide 27 by two | 13 | 1 |
| divide 13 by two | 6 | 1 |
| divide 6 by two | 3 | 0 |
| divide 3 by two | 1 | 1 |
| divide 1 by two | 0 | 1 |

Positive and Negative Numbers:

To denote the sign of a number (positive or negative), python adds an extra bit to the front. Thus, the first bit of any binary integer in python denotes the sign, with a 1 denoting a negative number, and a 0 denoting a positive number.

Irrational Numbers:

Since we have a finite number of transistors in a computer, we can only have finite precision for irrational numbers such as $\pi$. So, for a 64-bit computer, we can use 64 bits to represent the digits of an irrational number.

### Values

A value is a piece of data that is stored and manipulated in computer memory by a program. A value is also called an object in python. Every value has a type associated with it; some basic types are shown below:

int – integers, such as 123, 0, -11

float – real numbers (has a decimal point), such as 123.0, 44.1, 2.49

str – text, such as hello, goodbye

String values can contain almost any symbol and outputs the text contained within a set of single or double apostrophes. However, if the text requires single or double apostrophes, you will need to make sure that the text does not use the same type of apostrophe used to enclose the string text. Using the \ symbol, you can also tell python to ignore the single character afterwards and just treat it as part of the string text.

```
>>> print('monday')
monday
>>> print("monday")
monday
>>> print("it's monday")
it's monday
>>> print('it\'s monday')
it's monday
```

**Common Operations**

Addition – use the + sign to add two values

Subtraction – use the – sign to subtract one value from another

Multiplication – use the * symbol to multiply two numbers

Division – use / to divide one number by another

Modulo – use % to obtain the remainder of a division from one number by another

Exponentiation – use ** to calculate the exponent of one number by another

Negative – use – to make a value negative

Floor – use // to divide one number by another and then round down the answer to nearest integer

Division yields a float value, even if both numbers are integers, whereas floor division rounds down to the nearest whole number after the division and yields an integer value if both numbers are integers. Thus, if any of the numbers is a float value, the result will always be a float.

The modulo operator (%) finds the remainder of a division expression by finding the closest, smaller number that is divisible by the denominator (no remainder). For example, python solves 10%3 by finding the closest, smaller number to 10 that is divisible by 3, which in this case would be 9. The difference between 10 and 9 is 1, so 10%3 is equal to 1.

Additionally, in the case of python, the sign of the output for the modulo operator is determined by the sign of the denominator. So, -10%3 is equal to 2, since -12 is the closest, smaller number to -10 and the difference between -10 and -12 is 2. However, 10%-3 is equal to -2, and that's because we use the sign of the denominator.

The order of operations is the same for math (PEDMAS/BEDMAS). In case of ties, the operations are evaluated from left to right. The exception to this rule is that exponentiation associativity is from right to left (2**3**2 = 2**(3**2) = 512).

**Variables**

A variable refers to a specific value that is stored within the computer. A programmer defines a variable and can reference that variable using the name of the variable. Variables are created using the assignment operator, which is the equal sign (=). To make a variable, write a word or letter and set that equal to the information you would like to store. We can do many things with a variable without having to recompute it. Additionally, multiple variables can be assigned in one line.

```
>>> name, age, height = 'Bob', 18, 1.80
>>> print(name)
    Bob
>>> print(age, height)
    18 1.8
```

Variables can only start with a letter or the underscore symbol and cannot be keywords.

To use a variable, we can use a variable within a function (print(variable)), we can perform operations using a variable and a value (variable + value), or we can perform operations using two or more variables (variable + variable).

We can also change the value of a variable. If a variable is already defined, we can assign the variable a new value which replaces the old value (each variable can only have one value). The most recent variable assignment defines the variable's value.

```
>>> name = 'Bob'
>>> print(name)
    Bob
>>> name = 'Bobby'
>>> print(name)
    Bobby
```

We can also swap variable values by introducing a new variable. In the following example, x is equal to 5 and y is equal to 10, but to switch the values we cannot simply set x equal to y or vice versa because one of the values will be lost. Instead, we make a new variable and set one value equal to the new variable so that we can then swap the variable values without losing one of them.

```
>>> x = 5
>>> y = 10
>>> temp = x
>>> x = y
>>> y = temp
>>> print (x, y)
    10 5
```

**Expressions**

An expression is any combination of values, variables, operations, and functions. That means that assigning a value to a variable is an expression, just how combining values with operators is also an expression.

In Python, the equal sign does not indicate equality or equivalent expressions/values, instead, the equal sign symbolizes assignment. For example, x = x + 1 makes no sense in math because it is impossible for x to equal x +1, however, in Python, you are simply changing the value of x by adding 1.

```
>>> x = 1
>>> x = x + 1
>>> print (x)
    2
```

**Comments**

It is good practice to add notes in your code to describe what is going on. These notes are called comments, and they begin with the # symbol. Comments are ignored by Python (do not affect the code, only to help the user).

**Basic Terminology**

Shell and Prompt - The Python Shell is the interpreter that executes your Python programs, the prompt is the >>> which the computer displays to tell you it's ready for your instructions.

Statement - An instruction that a Python interpreter can execute.

Expression - An expression in Python is a combination of operators and operands that is interpreted by a Python interpreter to produce an output.

Operator and Operand - Operators are special symbols (such as the + or * signs) that designate that some sort of computation should be performed, the values that an operator acts on are called operands.

Types - Types refer to the classification of data (i.e., int, float, str).

Errors - Python returns an error when the code cannot be run (impossible statement or Python does not understand the code). Simply put, whenever a line of code does not make sense to Python or it cannot be run, an error will occur.

## 2. Strings

String values can be created using single or double quotes (quotes are not part of the value, they simply indicate the start and end of the string). Even if the string contains no values (known as an empty string), it is still a string. Strings denoted with single quotes can contain double quotes and vice versa. It is important to note that you must start and end each string with the same type of quotations.

The length of a string can be returned using the function len(). The length of a string is the number of characters including spaces contained within the string.

```
>>> print(len("it's monday"))
    11
>>> print(len(""))
    0
```

Strings can be concatenated using the + operator and repeated using the * operator.

```
>>> print("He"+"llo")
    Hello
>>> print("Hello"*3)
    HelloHelloHello
```

A string can be defined using a variable, and that string can be divided into sub-strings (part of a string). It is important to note that in python, 0 indicates the first value/digit, and for sub-strings, the interval is inclusive for the first digit but not for the last digit.

```
>>> x = "abcdefg"
>>> print(x)
    abcdefg
>>> print(x[0:3])
    abc
>>> print(x[:3])#shorthand
    abc
>>> print(x[1:3])
    bc
```

We can refer to specific characters in a string using indices which can also be used as values. If an index that is out of range (i.e., less characters than the index) is called, an index error will occur.

```
>>> x = "abcdefg"
>>> print(x[7])
    Traceback (most recent call last):
      Python Shell, prompt 24, line 1
    builtins.IndexError: string index out of range
```

Negative indices work in the opposite direction of positive indices, in that they count from right to left. Thus, the index -1 refers to the last character of a string and the index -2 refers to the second last character of a string.

```
>>> x = "abcdefg"
>>> print(x[-1])
    g
```

It is important to note that strings are immutable (cannot be modified, can only be replaced).

Basic String Functions and Methods:

The upper() and lower() methods change the characters to upper or lower case, respectively (can only be used on string types). As a method, upper() and lower() must be associated with an object (a string).

```
>>> print(x.upper(), x.lower())
    HELLO WORLD hello world
```

Use the “in” operator to check if an input or value is within another string.

```
>>> x = "Hello World"
>>> print("Hello" in x)
    True
```

Use the startswith() and endswith() methods to check if a string starts or ends with specific characters or strings.

```
>>> x = "Hello World"
>>> print(x.startswith("Hello"), x.endswith("World"))
    True True
```

## 3. Booleans and Logical Operators

A Boolean is a value type that evaluates to either true or false and is typically a comparison operator that compares two operators in some way. Boolean expressions belong to the bool type and are not strings. Boolean expressions only return the values “True” or “False”.

Earlier, it was stated that the equal sign (=) does not mean equal in python, rather, it represents assignment. Equality is denoted by the double equal sign (==), and this is one of many comparison operators. Some other common Boolean/comparison operators are shown below.

| Boolean Operator | True if: |
|---|---|
| x == y | x is equal to y |
| x > y | x is greater than y |
| x < y | x is less than y |
| x >= y | x is greater than or equal to y |
| x <= y | x is less than or equal to y |
| x != y | x does not equal y |

Comparison operators also work on strings, however, there are some rules. Firstly, characters in the strings must match exactly, including capitalization. Secondly, upper-case letters have smaller “values” than lower-case letters, but otherwise, earlier letters in the alphabet are smaller than those that come later in the alphabet. This is because Python uses the ASCII table for Unicode characters, and upper-case letters come before lower-case letters (lower value).

```
>>> print("Hello" == "Hello", "Hello" == "hello")
True False

>>> print("c" > "C", "c" > "d")
True False
```

Logical Operators:

Logical expressions can be more complex using logical operators such as “and”, “or”, and “not”.

The “and” operator combines two expressions, and both must be true to return True, otherwise it returns False.

Only one of two expressions needs to be true if the “or” operator is used to return True. If neither are true then it returns False.

The “not” operator simply results in the opposite answer (i.e., if originally it returned True, it would now return False.

If multiple logical operators are used, the order of precedence is left to right unless parentheses are used.

## 4. If Statements

The if statement is a condition that if true, will execute the code embedded in it. Otherwise, the code inside the if statement is skipped. To discern the code in an if statement, the code is indented. If statements begin with the keyword “if” followed by a condition/s.

```
if num > 0:
    print("The number is greater than 0")
```

If-else statements are an expansion of the if statement where only if the condition is not true, the code in the else section will execute. The “else” is not indented, but the code of the “else” section is.

```
if num > 0:
    print("The number is greater than 0")
else:
    print("The number is less than or equal to 0")
```

If statements can also be linked using “elif”. This is important because the if-elif-else statement will only execute the block of code corresponding to the first condition that is true (instead of multiple single if statements which may all execute if their conditions are met).

```
if num > 0:
    print("The number is greater than 0")
elif num > -10:
    print("The number is greater than -10 but less than 0")
elif num > -20:
    print("The number is greater than -20 but less than -10")
else:
    print("The number is less than or equal to -20")
```

If statements can also be nested within other if statements which is discerned by indentation.

## 5. Iteration

Iteration is the idea of repeating an activity multiple times and is also referred to as looping. In Python, three common ways to repeat are the while loop, for loop, and recursion.

### While Loop

The while loop repeats code if the condition remains true. Once the condition becomes false, the loop ends. A while loop begins with the keyword “while” followed by a condition.

```
while num < 10:
    num = num + 1
    print(num)
```

### For Loop

The for loop is specifically designed to count. The statement block is repeated until there is no more data in the object. A for loop begins with the keyword “for” followed by a variable that is counted within an object.

```
for number in range(10):
    print(number)
```

For loops are very useful for many functions, such as counting the number of times a value appears in a list or a letter appears in a string. The following code uses a for loop to count the number of times the letter L appears in the phrase “Hello World”.

```
word = "Hello World"
count = 0
for letter in word:
    if letter == "l":
        count = count + 1
print(count)
```

Range Function:

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number (does not include the end number specified in range).

```
>>> for num in range(10):
...     print(num)
...
0
1
2
3
4
5
6
7
8
9
```

The format of the range() function is range(start, end+1, step) where start is the starting value, end is the ending value, and step is the increment size.

Recursion:

Recursion is similar to a while loop but uses a function. It is important to be careful as recursion uses a lot of computer memory.

```
1   def factorial(x):
2       if x == 0:
3           return 1
4
5       answer = x * factorial(x-1)
6
7       return x
```

Essentially, recursion works by having a function call inside itself. Each call to the function creates a new instance (node) of “local” variables on the stack (run-time stack in RAM). This is why recursion uses a lot of computer memory.

Break and Continue:

The keyword “break” stops a loop before it is done executing its body. If a break is placed within a loop and it is executed, Python will exit the loop. To end the code for a specific iteration but still move to the next iteration, use “continue” instead of break.

## 6. Functions

A function can be thought of as a small program that has a specific purpose and syntax. For example, the print function only prints information to the screen, but the function itself must be spelled correctly, and the expression in the brackets must follow the correct syntax rules. When we use a function, we say that we “call” the function.

There are two types of functions, built-in functions which are already present in the coding language, and user-created functions which are functions that the programmer creates. Both types of functions follow the same syntax dictated by the coding language. All functions have three parts: a name, input value(s) that you put in the brackets, and a single output value returned as the answer. Inputs are also referred to as arguments and outputs are also referred to as return values.

Within functions, we can define variables, however, these variables can only be used in the function they are written in and are known as local variables. Global variables are variables outside of function definitions and can be used wherever in the code. This means when Python executes code outside of the function, the variables defined within a function cannot be accessed.

Print Function:

The function print() allows the user to print a value as text. Strings (str) need to be enclosed by double/single apostrophes while integers (int) and real numbers (float) do not. For print, the brackets denote what needs to be printed.

```
>>> print("Hello")
Hello
>>> print(1234)
1234
>>> print(123.4)
123.4
```

The print function can also print multiple values that can have different types, you simply separate them with a comma.

Input Function:

Using the input() function, we can ask the user to enter data from the keyboard, and we store the entered value in a variable.

```
>>> ID = input("Please input your ID: ")
Please input your ID: 1234
>>> print(ID)
1234
```

When the input() function is called, the Python program is paused. Once the user inputs information using the keyboard and presses the enter (return) key, the Python program resumes. It is important to note that the input() function always returns a string type, even if only numbers were inputted.

Value Type Conversion Functions:

Using the integer function int(), we can change the typing of a variable to the integer type. Similarly, using the str() function, we can change the typing of a variable to the string type, and the float() function does the same but to the float type.

User-Created Functions:

Just like built-in functions, invented functions are very useful. It allows for code to be reused and it makes the program easier to read and understand by organizing it.

To define a simple function, use the keyword “def”. The function requires a name, and if applicable, arguments. This first line is called the signature. Below the signature, add in statements to specify orders to the function, and make sure that the statements are indented. If an argument (input) is added, the argument can be defined for the function.

```
1  def function1():
2      print("Hello World")
3
4  def function2(x):
5      print(x * 2)
```

When defining a function, use the return function inside a function or method to send the function's result back to the caller (return also marks the end of the function but is not necessary for all functions). Then, when the function is called, the value of the function is defined by the value returned.

```
>>> def function2(x):
...     y = x * 2
...     return y
...
... print(function2("hello"))
...
   hellohello
```

Random Function:

To generate a random integer, we need to import the randint() function from the random library (a built-in library). Then, you specify the range for the random integer.

```
1  from random import randint
2
3  num = randint (0, 10)
4  print(num)
```

Function vs. Method:

A function is a “stand-alone” program that you can call from your program whereas a method is a function that “belongs” to an object. The dot operator is used to indicate when something belongs to an object.

To call a function: function_name(arguments)

To call a method: object.method_name(arguments)

## 7. Lists

A list is an ordered sequence of values defined by square brackets, []. Each value in the list is called an element and each element is an object. A list can contain no variables, variables of different value types, and even other lists. An empty list contains no elements.

To refer to a specific value in a list, use the list index, with 0 representing the first value in the list, and n-1 (where n is the number of values in a list) representing the last value in the list.

```
>>> list = [1, 'a', 2, 'b']
>>> print(list[0], list[2])
    1 2
```

Lists are mutable and a list index can be assigned a new value that can be of any value type. We can also use an expression for the list index. A negative index starts from the last index, with index [-1] being the last value in a list (as with strings). Using the “in” command, we can search for a value in a list.

The == operator checks if two lists are equal. Two lists are equal if they have the same number of elements, and the element at each index in list 1 is equal to the element at the same index in list 2.

```
>>> list1 = [1, 2, 3]
>>> list2 = [1, 2, 3]
>>> list3 = [1, 3, 2]
>>> print(list1 == list2, list1 == list3)
    True False
```

Lists can also be sliced in the same way as strings. If both the start and end indices are omitted, it is simply a copy of the whole list.

Append Method:

The append() method functions to add a new element to the end of the list (the + operator concatenates two lists to accomplish the same thing).

```
>>> list = [1, 2]
>>> list.append(3)
>>> print(list)
    [1, 2, 3]
```

Extension Method:

The extend() method appends all the elements of a list to the end of another. Note that the other list is unchanged.

```
>>> list1 = [1, 2]
>>> list2 = [3, 4]
>>> list1.extend(list2)
>>> print(list1)
    [1, 2, 3, 4]
```

Insert Method:

The insert() method adds an element to a list at a specific index, moving all other elements to the right and uses the format (index, value).

```
>>> list = [1, 2, 4]
>>> list.insert(2, 3)
>>> print(list)
    [1, 2, 3, 4]
```

Sort Method:

The sort() method arranges all the elements from lowest value to highest value.

```
>>> list = [1, 3, 4, 2]
>>> list.sort()
>>> print(list)
    [1, 2, 3, 4]
```

Sum Function:

To add up all the numbers in a list, we can use the sum() function to sum the values of a list (only int and float types).

```
>>> list = [5, 5, 10]
>>> print(sum(list))
    20
```

Pop Method:

The pop() method deletes an element from a list at a specific index. The pop() method modifies the list and returns the element that was removed.

```
>>> fruits = ["banana", "apple", "car"]
>>> not_fruits = fruits.pop(2)
>>> print(fruits)
    ['banana', 'apple']
>>> print(not_fruits)
    car
```

Alternatively, if you don’t need the removed value, you can use the del operator.

```
>>> fruits = ["banana", "apple", "car"]
>>> del fruits[2]
>>> print(fruits)
    ['banana', 'apple']
```

Remove Method:

If you know the element you want to remove but not the index, you can use the remove() method. The remove() method removes the first occurrence of the specified element.

```
>>> fruits = ["banana", "apple", "car"]
>>> fruits.remove("car")
>>> print(fruits)
    ['banana', 'apple']
```

Index Method:

The index() method returns the index of the first occurrence of a specified element in a list.

```
>>> names = ["bob", "henry", "bob"]
>>> print(names.index("bob"))
    0
```

Count Method:

The count() method returns the number of occurrences of a specific element in a list.

```
>>> names = ["bob", "henry", "bob"]
>>> print(names.count("bob"))
    2
```

More Methods and Functions for Lists:

list_name.reverse() reverses the order of the elements in a list.

len(list_name) returns the number of elements in the list.

min(list_name) returns the minimum value in the list.

max(list_name) returns the maximum value in the list.

**List of Lists**

A list can contain lists as elements. By providing a second index, we get an element from an inner list.

```
>>> student_grades = [[90, 90, 91], [100, 100, 99], [70, 80, 90]]
>>> print(student_grades[1][2])
    99
```

**Converting a String into a List**

The list() function changes a string into a list of characters.

```
1   name = "bob"
2   list1 = list(name)
3   print(list1)
4
5   #output: ['b', 'o', 'b']
```

Split Method:

The list() function breaks a string into individual letters. The split() method breaks a string into words.

```
name = "bob is cool"
list1 = name.split()
print(list1)

#output: ['bob', 'is', 'cool']
```

An optional argument for the split() method called the “delimiter” specifies which characters to use as word boundaries (if unspecified, python assumes the delimiter as a space). For example, we can specify the delimiter to be a comma instead.

```
groceries_list = "apples,milk,bread"
list1 = groceries_list.split(",")
print(list1)

#output: ['apples', 'milk', 'bread']
```

Join Method:

The join() method is the opposite of the split() method and it takes as an argument a list of strings and joins the elements. The join() method is a string method, so you have to call it on the delimiter.

```
list1 = ["bob", "is", "cool"]
delimiter = " "
phrase = delimiter.join(list1)
print(phrase)

#output: "bob is cool"
```

## 8. Dictionaries

A dictionary is a programmer defined mapping between a key and a value. Dictionary keys must be immutable (int, string, bool, float, etc.), so they cannot be lists or dictionaries. Keys must be unique, however, values do not.

A dictionary has the following format.

```
dictionary = {"key1": "value1", "key2": "value2"}
```

To add elements to a dictionary, specify a key and value pair.

```
dictionary = {} #empty dictionary
dictionary["key1"] = "value 1"
dictionary["key2"] = "value 2"
print(dictionary)

#output: {'key1': 'value 1', 'key2': 'value 2'}
```

If we use a key that already exists in the dictionary, the new value will overwrite the old value.

To delete an element from a dictionary, use the del command.

```
dictionary = {"key1": "value1", "key2": "value2"}
del dictionary["key1"]
print(dictionary)

#output: {'key2': 'value2'}
```

Values Method:

To check if a value is present in a dictionary, use the values() method.

```
dictionary = {"key1": "value1", "key2": "value2"}
print("value2" in dictionary.values())

#output: True
```

List Function and Dictionaries:

We can get a list of the keys in the dictionary using list().

Update Method:

To combine two dictionaries, we use the update() method.

```
dictionary1 = {"key1": "value1"}
dictionary2 = {"key2": "value2"}

dictionary1.update(dictionary2)
print(dictionary1)

#output: {'key1': 'value1', 'key2': 'value2'}
```

Dictionary Equality:

The == operator checks if two dictionaries are equal. Two dictionaries are equal if they have the same number of key-value pairs, and each key-value pair in each dictionary is also contained in the other. It is important to note that dictionaries have no order, so as long as both dictionaries have the same key-value pairs, they are equivalent, regardless of the order of the key-value pairs.

## 9. Tuples and Sets

Tuples are similar to lists except they are immutable, meaning you cannot add or remove elements. Tuples are denoted by at least 2 values within () brackets. Tuples must have more than 1 elements (if there is only 1 element, use a comma to add an empty element). Tuples can be indexed like a list, and strings and other data types can be turned into a tuple using the tuple() function. Negative indexing and slicing also applies to tuples.

```
1  tuple = ("value1", "value2")
2  print(tuple[1])
3
4  #output: 'value2'
```

A set is an unordered collection of unique values and a set cannot store duplicate values. A sequence of comma-separated values within {} denotes a set. The function set() will create an empty set.

If a sequence (list, tuple, string, etc.) is passed to the set() constructor, only unique items will be kept. Since sets are not ordered, the order in a list, tuple, etc. is lost. Only immutable objects can be put into a set.

```
1  list1 = ["value1", "value2", "value2"]
2  set1 = set(list1)
3  print(set1)
4
5  #output: {'value1', 'value2'}
```

The add() and remove() methods can be used to add or remove values in a set. The in operator can be used to check if a value is in a set.

Intersection Method:

The intersection() method returns a new set that contains the common values between 2 sets.

```
1  colors = {"red", "orange", "yellow", "blue"}
2  fruits = {"banana", "apple", "orange", "pear"}
3  print(colors.intersection(fruits))
4
5  #output: {'orange'}
```

Union Method:

The union() method returns a new set that contains all the unique values between 2 sets.

```
1  colors = {"red", "orange", "yellow", "blue"}
2  fruits = {"banana", "apple", "orange", "pear"}
3  print(colors.union(fruits))
4
5  #output: {'blue', 'red', 'banana', 'orange', 'yellow', 'pear', 'apple'}
```

Difference Method:

The difference() method returns a new set that contains values that are unique to the specific set (i.e., only in set A, not in set B).

```
colors = {"red", "orange", "yellow", "blue"}
fruits = {"banana", "apple", "orange", "pear"}
print(colors.difference(fruits))

#output: {'red', 'yellow', 'blue'}
```

## 10. NumPy

NumPy is the core library for scientific computing that must be imported. Importantly, NumPy provides a new data structure called an array. The array can be thought of as a more efficient list. A NumPy array is a multidimensional grid of values with rows and columns (vector: linear array, matrix: array of arrays). Unlike a regular list, all the values in the NumPy array must have the same type and the array's size cannot be changed after creation.

We can create a NumPy array from a list using the array() method. An array can be differentiated from a list because an array does not have commas separating the elements.

```
1  import numpy as np
2
3  array1 = np.array([1, 2, 3, 4])
4  print(array1)
5
6  #output: [1 2 3 4]
7  #note the lack of commas between elements in an array
```

We can also make NumPy arrays using the range() function and tuples.

```
1  import numpy as np
2
3  array1 = np.array(range(1, 5))
4  array2 = np.array((1, 2, 3, 4))
5  print(array1)
6  print(array2)
7
8  #output for both: [1 2 3 4]
```

Since arrays are multidimensional, the len() function does not work. Instead, the shape attribute is used to indicate an array's width (number of rows) and length (number of columns). The ndim attribute gives the number of dimensions (number of rows).

```
 1  import numpy as np
 2
 3  array1 = np.array([[1,2,3,4], [1,2,3,4]])
 4  print(array1)
 5  print(array1.ndim)
 6  print(array1.shape)
 7
 8  '''output:
 9  print(array1) = [[1 2 3 4]
10                  [1 2 3 4]]
11  print(array1.ndim) = 2
12  print(array1.shape) = (2, 4)
13  '''
```

Indexing an array is the same as indexing a list. While it is not possible to add or remove elements from arrays, the values of an array can be modified.

Zeros and Ones Method:

To create an array of zeroes with a defined shape, use the zeros() method. Similarly, to create an array of ones with a defined shape, use the ones() method. The default data type for zeros() and ones() is float. The full() method allows the user to choose the value. The random() method creates an array of a specified shape with random values between 0 and 1.

```
import numpy as np

print(np.zeros(3))

#output: [0. 0. 0.]
```

Arange Method:

Instead of range(),the arange() method creates a range as an array. Unlike range(), arange() can have float values.

```
import numpy as np

print(np.arange(0.0, 10.0, 2))

#output: [0. 2. 4. 6. 8.]
```

Linspace Method:

To create an array of evenly-spaced numbers within a range, it is better to use the linspace() method instead of arange().

```
import numpy as np

print(np.linspace(0, 10, num = 6))

#output: [0. 2. 4. 6. 8. 10.]
```

Arithmetic Operators:

Arithmetic operators on an array by a scalar value are applied to all elements of the array. This is known as broadcasting. This is different in lists as most arithmetic operators result in errors. Multiplication will cause the list to repeat.

```
import numpy as np

list1 = [1, 2, 3]
array1 = np.array([1, 2, 3])

print(list1 * 2)
print(array1 * 2)

'''output:
print(list1 * 2) = [1, 2, 3, 1, 2, 3]
print(array1 * 2) = [2 4 6]
'''
```

Math operations between arrays are done on an element-by-element basis. This is known as a vectorized operation. Essentially, the corresponding elements in each row have a specific arithmetic operation done on them.

```
import numpy as np

array1 = np.array([2, 4, 6])
array2 = np.array([1, 2, 3])

print(array1 + array2)

#output: [3, 6, 9]
```

Matrix Operations:

Math operators can be used on matrices as well.

```
import numpy as np

array1 = np.array([[2, 4, 6], [1, 2, 3]])

print(array1 + 2)

#output: [[4 6 8]
#         [3 4 5]]
```

Matrix multiplication is possible if two matrices have the same shape, where each value is multiplied by its corresponding value in the other matrix.

```
import numpy as np

array1 = np.array([[2, 4, 6], [1, 2, 3]])
array2 = np.array([[2, 4, 6], [1, 2, 3]])

print(array1 * array2)

#output: [[ 4 16 36]
#         [ 1  4  9]]
```

If the matrices are not the same shape, the dot() method multiplies matrices according to matrix multiplication in math.

```
import numpy as np

array1 = np.array([[2, 4, 6], [1, 2, 3]])
array2 = np.array([2, 4, 6])

print(array1.dot(array2))

#output: [56, 28]
```

To slice a matrix, use the notation matrix_name[(row, column), (row, column)], where the row and column follow slice notation ([start:end:increment]).

For example, to get all the elements in the first row use: matrix_name[0,:]

## 11. Object Oriented Programming

So far, only procedural programming has been described. Procedural programming divides a main program into functions. Object oriented programming (OOP) divides a main program into pieces called “objects”. Within each object is data and methods that belong to a single theme (e.g., the math module has a theme in that it contains methods related to math operators).

OOP has new terminology which is described below:

Class: the keyword to declare code belonging to an object

Instance: the action of creating an object from a class

Method: the functions that belong to a class

Attributes: the variables that belong to a class

Constructor: the initial or default values of an object (__init__)

The format of a class is shown below:

```
class Student: #class has theme of student
    def __init__(self, name, faculty): #constructor
        self.name = name #attribute name
        self.faculty = faculty #attribute faculty

    def is_science(self, faculty): #method
        if self.faculty == "science":
            return True
        else:
            return False

Bob = Student("Bob", "science") #instance
```

__eq__ method:

If we compare two objects using the == operator, by default Python will just compare them by their IDs. However, by implementing the method __eq__ in a class, the default comparison behavior is overridden and Python compares the values of variables instead of the IDs.

## 12. Random Numbers

Random numbers are highly applicable in science and engineering computations. Random numbers that are generated by a computer are pseudorandom as they are based on deterministic algorithms. These algorithms can follow different distributions.

Two common distributions are the uniform distribution (any number in the range is equally likely), and normal distribution (midpoint numbers are more likely).

Since computer randomness is based on deterministic algorithms, it is possible to predict the next value. Specifically, functions that create random numbers require a seed value (starting value). This number is used as the next seed value and this continues every time a random number is generated. Importantly, given the same seed value, the same sequence of “random” numbers will be generated.

The random imported library is used to generate random numbers. From the random module, the random(), randint(), and randomn() methods generate random values.

Random Method:

The random() method generates random, uniformally distributed float values from 0 to 1 (not including 1).

```
1  import random
2
3  print(random.random())
4
5  #output: any value [0, 1)
```

Randint Method:

The randint() method generates random, uniformally distributed integer values between 2 specified values (inclusive).

```
1  import random
2
3  print(random.randint(1, 10))
4
5  #output: any value from 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

Randomn Method:

The randomn() method generates random, normally distributed float values from 0 to 1 (not including 1).

```
1  import random
2
3  print(random.randomn())
4
5  #output: any value [0, 1)
```

Choice Method:

The choice() method returns a random element from a given sequence.

```
import random

names = ["bob", "emma", "jim", "grace"]

print(random.choice(names))

#output: any of "bob", "emma", "jim", or "grace"
```

Sample Method:

The sample() method returns a specified number of random elements of the sequence. The sample() method will not return two of the same element unless there are duplicate elements. This is analogous to picking numbers out of a hat, where once picked, that number no longer exists in the hat.

```
import random

names = ["bob", "emma", "jim", "grace"]

print(random.sample(names, 3))

#output: 3 unique names from the list "names"
```