Python Data Types

- Expressions, Variables, and Assignments
- Strings
- Lists and Tuples
- Objects and Classes

Algebraic expressions

The Python interactive shell can be used to evaluate algebraic expressions

`14//3` is the quotient when 14 is divided by 3 and `14%3` is the remainder

`2**3` is 2 to the 3rd power

`abs()`, `min()`, and `max()` are functions

- `abs()` takes a number as input and returns its absolute value
- `min()` (resp., `max()`) take an arbitrary number of inputs and return the "smallest" (resp., "largest") among them

```
>>> 2 + 3
5
>>> 7 - 5
2
>>> 2*(3+1)
8
>>> 5/2
2.5
>>> 5//2
2
>>> 14//3
4
>>> 14%3
2
>>> 2**3
8
>>> abs(-3.2)
3.2
>>> min(23,41,15,24)
15
>>> max(23,41,15,24)
41
```

Algebraic expressions

Write Python algebraic expressions corresponding to the following statements:

(a) The sum of the first five positive integers
(b) The average age of Sara (age 23), Mark (age 19), and Fatima (age 31)
(c) The number of times 73 goes into 403
(d) The remainder when 403 is divided by 73
(e) 2 to the 10th power
(f) The absolute value of the difference between Sara's height (54 inches) and Mark's height (57 inches)
(g) The lowest price among the following prices: $34.99, $29.95, and $31.50

Boolean expressions

In addition to algebraic expressions, Python can evaluate Boolean expressions

- Boolean expressions evaluate to `True` or `False`
- Boolean expressions often involve comparison operators
  <, >, ==, !=, <=, and >=

```
>>> 2 < 3
True
>>> 2 > 3
False
>>> type(2>3)
<class 'bool'>
>>> 2 == 3
False
>>> 2 != 3
True
>>> 2 <= 3
True
>>> 2 >= 3
False
>>> 2+4 == 2*(9/3)
True
```

In a an expression containing algebraic and comparison operators:

- Algebraic operators are evaluated first
- Comparison operators are evaluated next

Boolean expressions

Translate the following statements into Python Boolean expressions and evaluate them:

(a) The sum of 2 and 2 is less than 4.
(b) The value of `7 // 3` is equal to `1 + 1`.
(c) The sum of 3 squared and 4 squared is equal to 25.
(d) The sum of 2, 4, and 6 is greater than 12.
(e) 1387 is divisible by 19.
(f) 31 is even. (Hint: what does the remainder when you divide by 2 tell you?)
(g) The lowest price among $34.99, $29.95, and $31.50 is less than $30.00.

BASIC LOGIC OPERATIONS

1. NOT Operation:-

The NOT operation is a unary logic operator.
It changes one logic to opposite logic.

When:
Input = True, output = False
Input = False, output = True

Truth Table

P	NOT P
True	False
False	True

BASIC LOGIC OPERATIONS

2. AND Operation :-

AND is a Binary Operator

The AND operation produces a True only if all the inputs are True.

If any of the input is False, the output is False.

P	Q	P AND Q
False	False	False
False	True	False
True	False	False
True	True	True

BASIC LOGIC OPERATIONS

3. OR Operation :-

OR is a Binary Operator

The OR operation produces a True even if any one of the inputs is True.

OR output is only False if all its inputs are False.

P	Q	P OR Q
False	False	False
False	True	True
True	False	True
True	True	True

Boolean operators

In addition to algebraic expressions, Python can evaluate Boolean expressions

- Boolean expressions evaluate to `True` or `False`
- Boolean expressions may include Boolean operators `and`, `or`, and `not`

```
>>> 2<3 and 3<4
True
>>> 4==5 and 3<4
False
>>> False and True
False
>>> True and True
True
>>> 4==5 or 3<4
True
>>> False or True
True
>>> False or False
False
>>> not(3<4)
False
>>> not(True)
False
>>> not(False)
True
>>> 4+1==5 or 4-1<4
True
```

In a an expression containing algebraic, comparison, and Boolean operators:

- Algebraic operators are evaluated first
- Comparison operators are evaluated next
- Boolean operators are evaluated last

Exercise

Translate the following into Python algebraic or Boolean expressions and then evaluate them:

a) The difference between Annie's age (25) and Ellie's (21)
b) The total of $14.99, $27.95, and $19.83
c) The area of a rectangle of length 20 and width 15
d) 2 to the 10th power
e) The minimum of 3, 1, 8, -2, 5, -3, and 0
f) 3 equals 4-2
g) The value of 17//5 is 3
h) The value of 17%5 is 3
i) 284 is even
j) 284 is even and 284 is divisible by 3
k) 284 is even or 284 is divisible by 3

```
>>> 25 - 21
4
>>> 14.99 + 27.95 + 19.83
62.769999999999996
>>> 20*15
300
>>> 2**10
1024
>>> min(3, 1, 8, -2, 5, -3, 0)
-3
>>> 3 == 4-2
False
>>> 17//5 == 3
True
>>> 17%5 == 3
False
>>> 284%2 == 0
True
>>> 284%2 == 0 and 284%3 == 0
False
>>> 284%2 == 0 or 284%3 == 0
True
```

Variables and assignments

Just as in algebra, a value can be assigned to a variable, such as x

When variable x appears inside an expression, it evaluates to its assigned value

A variable (name) does not exist until it is assigned

The assignment statement has the format

```
<variable> = <expression>
```

<expression> is evaluated first, and the resulting value is assigned to variable <variable>

```
>>> x = 3
>>> x
3
>>> 4*x
16
>>> y
Traceback (most recent call
last):
  File "<pyshell#59>", line
1, in <module>
    y
NameError: name 'y' is not
defined
>>> y = 4*x
>>> y
16
```


Naming rules

(Variable) names can contain these characters:

- a through z
- A through Z
- the underscore character _
- digits 0 through 9

Names cannot start with a digit though

For a multiple-word name, use

- either the underscore as the delimiter
- or *camelCase* capitalization

Short and meaningful names are ideal

```
>>> x = 4
>>> x
4
>>> x = 7
>>> x
4
```

Strings

In addition to number and Boolean values, Python support string values

`'Hello, World!'`

A string value is represented as a sequence of characters enclosed within quotes

A string value can be assigned to a variable

String values can be manipulated using string operators and functions

```
>>> 'Hello, World!'
'Hello, World!'
>>> s = 'rock'
>>> t = 'climbing'
>>>
```


String operators

Usage	Explanation
`x in s`	`x` is a substring of `s`
`x not in s`	`x` is not a substring of `s`
`s + t`	Concatenation of `s` and `t`
`s * n, n * s`	Concatenation of `n` copies of `s`
`s[i]`	Character at index `i` of `s`
`len(s)`	(function) Length of string `s`

To view all operators, use the `help()` tool

```
>> help(str)
Help on class str in module builtins:

class str(object)
 |  str(string[, encoding[, errors]]) -> str
...
```

```
>>> s = 'rock'
>>> t = 'climbing'
>>> s == 'rock'
True
>>> s != t
True
>>> s < t
False
>>> s > t
True
>>> s + t
'rockclimbing'
>>> s + ' ' + t
'rock climbing'
>>> 5 * s
'rockrockrockrockrock'
>>> 30 * '_'
'______________________________'
>>> 5 * '_' + s
'_____rock'
>>> 5 * ('_' + s)
'_rock_rock_rock_rock_rock'
>>> 'o' in s
True
>>> 'o' in t
False
>>> 'bi' in t
True
>>> len(t)
8
```

Exercise

Write Python expressions involving strings `s1`, `s2`, and `s3` that correspond to:

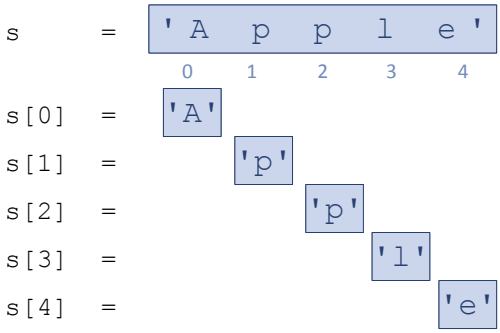
a) `'ll'` appears in `s3`
b) the blank space does not appear in `s1`
c) the concatenation of `s1`, `s2`, and `s3`
d) the blank space appears in the concatenation of `s1`, `s2`, and `s3`
e) the concatenation of 10 copies of `s3`
f) the total number of characters in the concatenation of `s1`, `s2`, and `s3`

```
>>> s1
'good'
>>> s2
'bad'
>>> s3
'silly'
>>> 'll' in s3
True
>>> ' ' not in s1
True
>>> s1 + s2 + s3
'goodbadsilly'
>>> ' ' in s1 + s2 + s3
False
>>> 10*s3
'sillysillysillysillysillysillysillysillysillysillysilly'
>>> len(s1+s2+s3)
12
>>>
```


Index and indexing operator

The index of an item in a sequence is its position with respect to the first item

- **The first item has index 0,**
- **The second has index 1,**
- The third has index 2, …



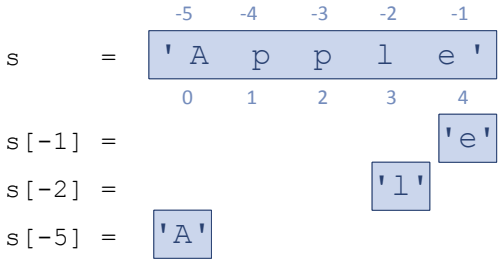
The indexing operator `[]` takes a nonnegative index `i` and returns a string consisting of the single character at index `i`

```
>>> s = 'Apple'
>>> s[0]
'A'
>>> s[1]
'p'
>>> s[4]
'e'
```

Negative index

A negative index is used to specify a position with respect to the "end"

- The last item has index -1,
- The second to last item has index -2,
- The third to last item has index -3, …




```
>>> s = 'Apple'
>>> s[-1]
'e'
>>> s[-2]
'l'
>>> s[-5]
'A'
```


Exercise

String `s` is defined to be

```
'abcdefgh'
```

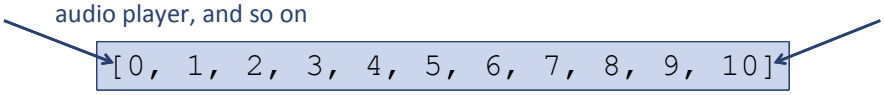
Write expressions using `s` and the indexing operator `[]` that return the following strings:

a) `'a'`
b) `'c'`
c) `'h'`
d) `'f'`

```
>>> s = 'abcdefgh'
>>> s[0]
'a'
>>> s[2]
'c'
>>> s[7]
'h'
>>> s[-1]
'h'
>>> s[-3]
'f'
>>>
```

Lists

In many situations we organize data into a list: a shopping list, a list of courses, a list of contacts on your cell phone, a list of songs in your audio player, and so on



In Python a comma-separated sequence of items enclosed within square brackets

The items can be numbers, strings, and even other lists

```
>>> pets = ['ant', 'bat', 'cod', 'dog', 'elk']
>>> lst = [0, 1, 'two', 'three', [4, 'five']]
>>> nums = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>>
```


List operators and functions

Like strings, lists can be manipulated with operators and functions

Usage	Explanation
`x in lst`	`x` is an item of `lst`
`x not in lst`	`x` is not an item of `lst`
`lst + lstB`	Concatenation of `lst` and `lstB`
`lst*n, n*lst`	Concatenation of `n` copies of `lst`
`lst[i]`	Item at index `i` of `lst`
`len(lst)`	Number of items in `lst`
`min(lst)`	Minimum item in `lst`
`max(lst)`	Maximum item in `lst`
`sum(lst)`	Sum of items in `lst`

```
>>> lst = [1, 2, 3]
>>> lstB = [0, 4]
>>> 4 in lst
False
>>> 4 not in lst
True
>>> lst + lstB
[1, 2, 3, 0, 4]
>>> 2*lst
[1, 2, 3, 1, 2, 3]
>>> lst[0]
1
>>> lst[1]
2
>>> lst[-1]
3
>>> len(lst)
3
>>> min(lst)
1
>>> max(lst)
3
>>> sum(lst)
6
>>> help(list
...
```

Lists are mutable, strings are not

Lists can be modified; they are said to be mutable

```
pets = ['ant', 'bat', 'cow', 'dog', 'elk']
```

Strings can't be modified; they are said to be immutable

```
pet = 'cod'
```

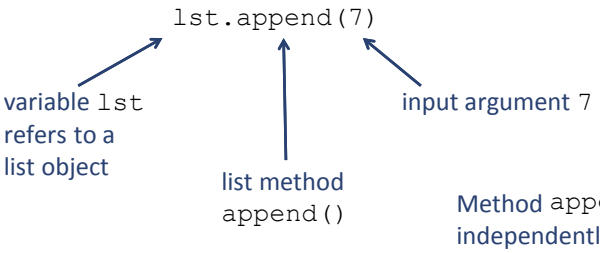
```
>>> pets = ['ant', 'bat', 'cod', 'dog', 'elk']
>>> pets[2] = 'cow'
>>> pets
['ant', 'bat', 'cow', 'dog', 'elk']
>>> pet = 'cod'
>>> pet[2] = 'w'
Traceback (most recent call last):
  File "<pyshell#155>", line 1, in <module>
    pet[2] = 'w'
TypeError: 'str' object does not support item assignment
>>>
```


Lists methods

`len()` and `sum()` are examples of functions that can be called with a list input argument; they can also be called on other type of input argument(s)

There are also functions that are called on a list; such functions are called list methods

```
>>> lst = [1, 2, 3]
>>> len(lst)
3
>>> sum(lst)
6
>>> lst.append(7)
>>> lst
[1, 2, 3, 7]
>>>
```



Method `append()` can't be called independently; it must be called on some list object

Lists methods

Usage	Explanation
`lst.append(item)`	adds `item` to the end of `lst`
`lst.count(item)`	returns the number of times `item` occurs in `lst`
`lst.index(item)`	Returns index of (first occurrence of) `item` in `lst`
`lst.pop()`	Removes and returns the last item in `lst`
`lst.remove(item)`	Removes (the first occurrence of) `item` from `lst`
`lst.reverse(item)`	Reverses the order of items in `lst`
`lst.sort(item)`	Sorts the items of `lst` in increasing order

Methods `append()`, `remove()`, `reverse()`, and `sort()` do not return any value; they, along with method `pop()`, modify list `lst`

```
>>> lst = [1, 2, 3]
>>> lst.append(7)
>>> lst.append(3)
>>> lst
[1, 2, 3, 7, 3]
>>> lst.count(3)
2
>>> lst.remove(2)
>>> lst
[1, 3, 7, 3]
>>> lst.reverse()
>>> lst
[3, 7, 3, 1]
>>> lst.index(3)
0
>>> lst.sort()
>>> lst
[1, 3, 3, 7]
>>> lst.remove(3)
>>> lst
[1, 3, 7]
>>> lst.pop()
7
>>> lst
[1, 3]
```


Exercise

List `lst` is a list of prices for a pair of boots at different online retailers

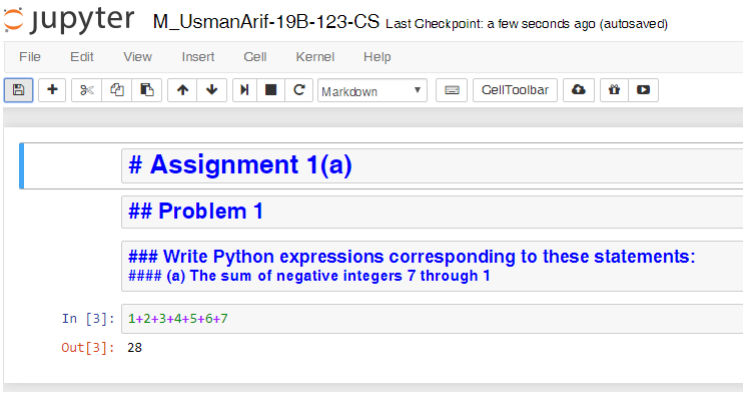
a) You found another retailer selling the boots for $160.00; add this price to list `lst`
b) Compute the number of retailers selling the boots for $160.00
c) Find the minimum price in `lst`
d) Using c), find the index of the minimum price in list `lst`
e) Using c) remove the minimum price from list `lst`
f) Sort list `lst` in increasing order

```
>>> lst = [159.99, 160.00, 205.95,
128.83, 175.49]
>>> lst.append(160.00)
>>> lst.count(160.00)
2
>>> min(lst)
128.83
>>> lst.index(128.83)
3
>>> lst.remove(128.83)
>>> lst
[159.99, 160.0, 205.95, 175.49, 160.0]
>>> lst.sort()
>>> lst
[159.99, 160.0, 160.0, 175.49, 205.95]
>>>
```

Assignment 1(a)

1. Will be uploaded on Shared folder today (28th Oct 2019)
2. The assignment is to be attempted in Jupyter Notebook as a single Notebook and submitted through email
3. Submission of assignment 1(a) is due at Midnight of the 3rd of November 2019.
4. The notebook must be saved using your name and Roll number (e.g. M_UsmanArif-19B-123-CS).
5. All the parts of the assignment must be attempted in a single Notebook and multiple copies will not be accepted as submission.
6. Any submission not following the above stated rules will not be accepted or marked

Assignment 1(a)



Built-in class `tuple`

The class `tuple` is the same as class list … except that it is immutable

```
>>> lst = ['one', 'two', 3]
>>> lst[2]
3
>>> lst[2] = 'three'
>>> lst
['one', 'two', 'three']
>>> tpl = ('one', 'two', 3)
>>> tpl
('one', 'two', 3)
>>> tpl[2]
3
>>> tpl[2] = 'three'
Traceback (most recent call last):
  File "<pyshell#131>", line 1, in <module>
    tpl[2] = 'three'
TypeError: 'tuple' object does not support item assignment
>>>
```

Why do we need it? Sometimes, we need to have an "immutable list".

Tuple operators

Like strings and lists, tuples can be manipulated with operators and functions

Usage	Explanation
`x in tpl`	`x` is an item of `tpl`
`x not in tpl`	`x` is not an item of `tpl`
`tpl + tpl2`	Concatenation of `tpl` and `tpl2`
`tpl*n, n*tpl`	Concatenation of `n` copies of `tpl`
`tpl[i]`	Item at index `i` of `tpl`
`len(tpl)`	Number of items in `tpl`
`min(tpl)`	Minimum item in `tpl`
`max(tpl)`	Maximum item in `tpl`
`sum(tpl)`	Sum of items in `tpl`

```
>>> tpl = (1,2,3,4)
>>> tpl2 = (0,6)
>>> 4 in tpl
True
>>> 4 in tpl2
False
>>> tpl + tpl2
(1, 2, 3, 4, 0, 6)
>>> tpl = tpl + tpl2
>>> tpl
(1, 2, 3, 4, 0, 6)
>>> tpl[1]
2
>>> tpl[1] = 0
Traceback (most
tpl[1] = 0
TypeError: 'tuple'
object does not
support item
assignment
>>> len(tpl)
7
>>> min(tpl)
0
>>> max(tpl)
6
>>> sum(tpl)
21
```