CLASS 11:
PYTHON DATA TYPES

ENGR 102 – Introduction to Engineering

Data Types

Assignment of Variables

Console 1/A

```
In [28]: a = 2

In [29]: b = 4.5

In [30]:
```

Name	Type	Size	Value
a	int	1	2
b	float	1	4.5

- Can define variables and assign values
  - Within a script
  - In the console
- Can then operate on those variables
- Variables appear in variable explorer

Variable Declaration

- In Python, it isn't necessary to declare a variable before using it, e.g.:

```
a = 7.4039
```

- Declaration occurs automatically upon assignment
- This differs from many other languages, e.g. in C:

```
float a;
a = 7.4039;
```

or

```
float a = 7.4039;
```

Variable Names

- Variable names must ***start with a letter*** or ***underscore***
- Names may contain ***letters***, ***numbers***, and ***underscore*** characters
  - ***No spaces***
- Some examples:

Allowed	Not allowed
A	Var 3
var1	4x_a
x_2_a	data file name
Avg_price	%pop #

Variable Names

- Names are ***case sensitive***
  - For example, all three are different:
    - name_1
    - Name_1
    - NAME_1
- Cannot use Python ***keywords***
  - E.g., for, if, def, True, etc.
- Don't name variables with names of ***built-in functions***
  - Can be done, but that function will become unavailable

- ***Preferred variable naming convention***:
  - All lowercase
  - Separate multiple words with an underscore

Variable Declaration – Dynamic Typing

- Python variables are of can be different ***types***, e.g.:
  - Integer: int
  - Floating-point number: float
  - Alpha-numeric string: str
- Python is ***dynamically typed***
  - Don't need to assign type when defining a variable
  - Python ***interpreter determines type*** at runtime

Console 1/A

```
In [22]: a = 2

In [23]: b = 3

In [24]: c = a/b

In [25]: d = 2.0

In [26]: greeting = 'Hello'
```

Name	Type	Size	Value
a	int	1	2
b	int	1	3
c	float	1	0.6666666666666666
d	float	1	2.0
greeting	str	5	Hello

Fundamental Python Data Types

- Python supports many different numeric and non-numeric ***data types***, for example
- **Numeric types**
  - `int`
  - `float`
  - `complex`
- **Non-numeric types**
  - `str`
  - `list`
  - `tuple`
  - `set`
  - `dict`
  - `bool`
- We'll introduce each of these types now, but will learn more about them throughout the course

Mutable vs. Immutable Data Types

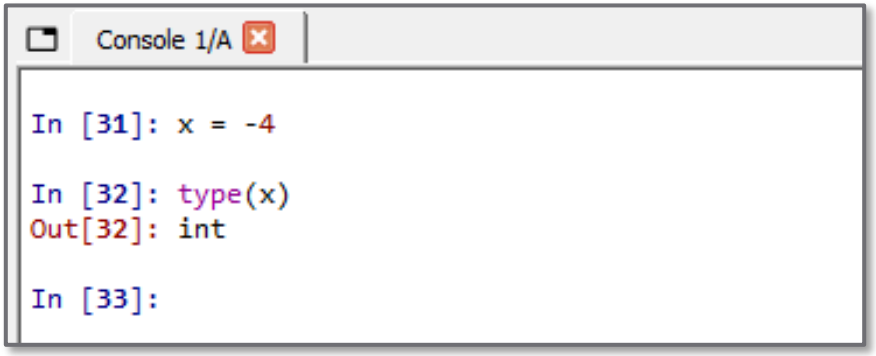
- Data objects of all types are values stored at specific locations in a computer's memory
- All data types fall into one of two categories:
  - ***Immutable***
    - Values cannot be modified after the variable is created in memory
      - Numbers – `int`, `float`, `complex`
      - Strings – `str`
      - Tuples – `tuple`
  - ***Mutable***
    - Values can be modified after variable creation
    - Can add, delete, insert, and rearrange items in a mutable sequence
      - Lists – `list`
      - Dictionaries – `dict`
      - Sets – `set`

Data Types – `int`

- ***Integers***
  - Zero, positive, and negative ***whole numbers***

```
>>> a = 7
>>> x = -4
>>> N = 0
```

- If you assign a whole-number value to a variable, it will automatically be cast as an `int`

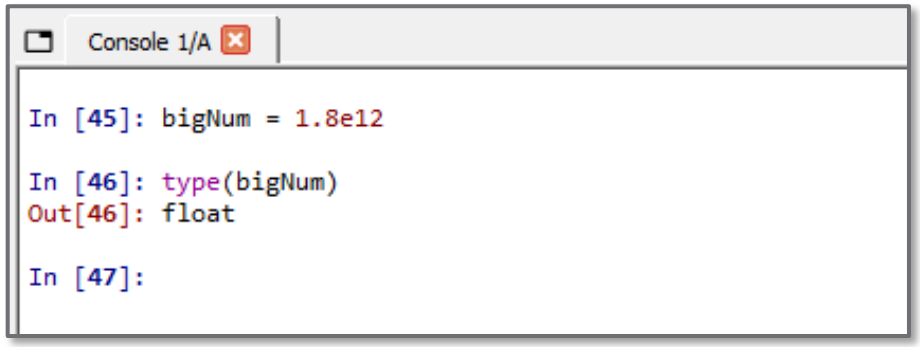


Data Types – `float`

- ***Floating point numbers***
  - Positive, and negative ***non-whole numbers***

```
>>> a = 2.71
>>> x = -4.5
>>> bigNum = 1.8e12
>>> smallNum = 6.4E-9
```

- If you assign a non-whole-number value to a variable, it will automatically be cast as a `float`



Scientific Notation

- Use ***scientific notation*** to represent very large or very small floating-point numbers, e.g.:

$$1.58 \times 10^{-9}$$

- Very bad practice to type a lot of zeros – ***never do this***:

$$0.00000000158$$

  - Difficult to read, and much too easy to miscount zeros

- In Python use e or E for $\times 10^{x}$, e.g.:

```
x = 1.58e-9

x = 1.58E-9
```

- Don't confuse with the exponential function e^{x} (i.e. 2.718^{x})

Data Types – complex

- ***Complex numbers***
  - Numbers with ***real*** and ***imaginary parts***

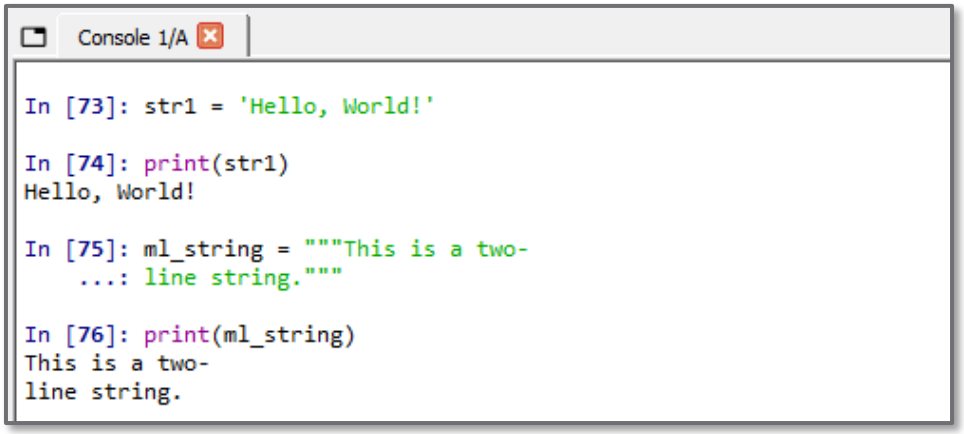
```
>>> z = 3 + 2j
>>> b = -4.5 + 6j
>>> V = 105 - 18.6j
```

- j is the imaginary unit
  - j = $\sqrt{-1}$

Data Types – str

- ***Strings***
  - Sequences of ***alpha-numeric characters***
  - Enclosed in single, double, or triple quotes

```
>>> str_1 = 'Hello, World!'
>>> Name = "John Doe"
>>> ml_string = '''Multi-line strings
are enclosed in
triple quotes.'''
```



Data Types – `str` – Escape Characters

- ***Escape characters***
  - Allows you to insert special characters in strings
  - Backslash, \, followed by a special character

Escape Character	Result
\'	Single quote
\"	Double quote
\\	Backslash
\n	New line
\t	Tab

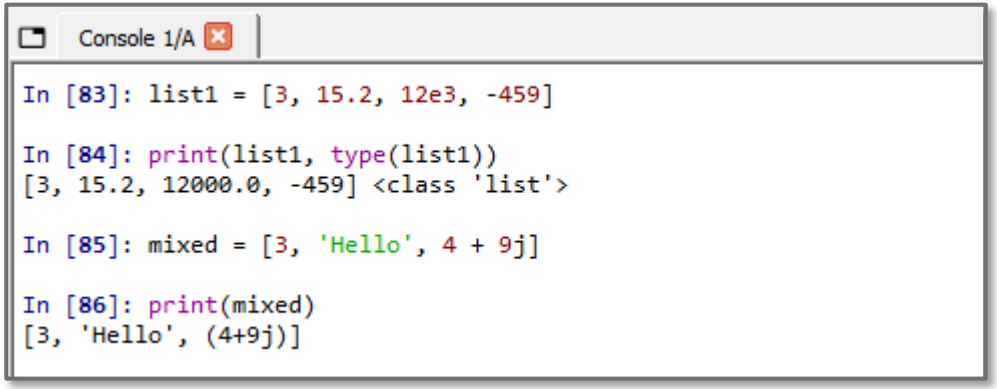
Console 1/A

```
In [403]: print('He said, \'hello!\'')
He said, 'hello!'

In [404]: print("He said, \"hello!\"")
He said, "hello!"

In [405]: print('C:\\Program Files\\Microsoft')
C:\Program Files\Microsoft

In [406]: print('Put this on one line\nand this on another.')
Put this on one line
and this on another.

In [407]: print('Separate\twith\ttabs.')
Separate	with	tabs.

In [408]:
```

Data Types – `list`

- ***Lists***
  - Ordered, mutable collections of one or more different data types
  - Enclosed in square brackets, [ ], separated by commas

```
>>> list1 = [3, 15.2, 12e3, -459]
>>> names = ['Jane', ‘Bob', 'Sally']
>>> mixed = [3, 'Hello', 4 + 9j]
```



Data Types – `tuple`

- ***Tuples***
  - Ordered, immutable collections of one or more different data types
  - Like a list, but immutable
  - Enclosed in parentheses, ( ), separated by commas

```
>>> tup1 = (3, 15.2, 12e3, -459)
>>> names = ('Jane', 'Bob', 'Sally')
>>> mixtup = (3, 'Hello', 4 + 9j)
```

```
Console 1/A

In [87]: tup1 = (3, 15.2, 12e3, -459)

In [88]: print(tup1)
(3, 15.2, 12000.0, -459)

In [89]: mixtup = (3, 'Hello', 4 + 9j)

In [90]: print(mixtup, type(mixtup))
(3, 'Hello', (4+9j)) <class 'tuple'>
```

Data Types – set

- ***Sets***
  - Unordered, mutable collections of one or more different data types
  - Enclosed in curly brackets, { }, separated by commas
  - Sets do not store duplicate objects
  - Suitable for mathematical set operations, e.g., union, intersection, difference, etc.

```
>>> numset = {3, 15.2, 12e3, -459}
>>> names = {'Jane', 'Bob', 'Sally'}
>>> set3 = {3, 'Hello', 4 + 9j}
```

```
Console 1/A
In [91]: numset = {3, 15.2, 12e3, -459}

In [92]: numset
Out[92]: {-459, 3, 15.2, 12000.0}

In [93]: names = {'Jane', 'Bob', 'Sally', 'Bob'}

In [94]: print(names, type(names))
{'Jane', 'Sally', 'Bob'} <class 'set'>
```

Data Types – dict

- ***Dictionaries***
  - Ordered, mutable collections of data stored as ***key:value*** pairs
  - Enclosed in curly brackets, { }
  - Keys and values separated by colons
  - Key:value pairs separated by commas
  - Duplicate keys are not allowed

```
>>> person1 = {'Name':, 'Joe', 'Age':, 32, 'Hair':,
'brown', 'Eyes':, 'green'}

>>> capitals = {'OR':, 'Salem', 'WA':, 'Olympia',
'CA':, 'Sacremento', 'ID':, 'Boise}
```

```
Console 1/A

In [118]: capitals = {'OR':'Salem','WA':'Olympia','CA':'Sacramento','ID':'Boise'}

In [119]: print(capitals,type(capitals))
{'OR': 'Salem', 'WA': 'Olympia', 'CA': 'Sacramento', 'ID': 'Boise'} <class 'dict'>

In [120]: capitals['OR']
Out[120]: 'Salem'
```

Data Types – bool

- ***Booleans***
  - One of two *logical* values: **`True`** or **`False`**
  - Often the result of a *logical expression*, e.g., `a > b`
  - Any value can be cast as a Boolean using the `bool()` function
    - True:
      - Non-zero numbers
      - Non-empty strings, lists, tuples, sets, or dictionaries
    - False:
      - Zero
      - Empty strings, lists, tuples, sets, or dictionaries

Console 1/A

```
In [128]: a = 4

In [129]: b = 8

In [130]: c = (b > a)

In [131]: print(c, type(c))
True <class 'bool'>

In [132]: bool(a)
Out[132]: True
```