Lecture 1-3

Basic Data Types and Variables

Week 1 Friday

Miles Chen, PhD

Adapted from *Think Python* by Allen B. Downey and *A Whirlwind Tour of Python* by Jake VanderPlas

Values and Types

There are different types of data in **base Python**. Other important data types also exist, but only after loading libraries like NumPy and Pandas. We will first begin with those in base.

The most commonly used ones will be:

- `str` - strings: for text data
- `int` - integers
- `float` - floats for numbers with decimal values
- `bool` - boolean: True or False
- `NoneType` - The reserved name and value `None` is used to indicate Null values

Python has important data structures that we will cover later, including:

- sequences: `list` `tuple` and `range`
- mappings: `dict`
- sets: `set`

- binary: `bytes`

```
In [1]: type("2")
```

```
Out[1]: str
```

```
In [2]: type(2)
```

```
Out[2]: int
```

```
In [3]: type(2.0)
```

```
Out[3]: float
```

```
In [4]: type(True)
```

```
Out[4]: bool
```

```
In [5]: type(None)
```

```
Out[5]: NoneType
```

Math operations in Python

Base Python has only a few math operations

- `x + y` sum of x and y.
- `x * y` multiplication of x and y.
- `x - y` difference of x and y.
- `x / y` division of x by y.
- `x // y` integer floor division of x by y.
- `x % y` integer remainder of `x//y`
- `x ** y` x to the power of y
- `abs(x)` absolute value of x

Adding integers together results in an integer

In [6]:
```
x = 10
y = 5
print(type(x))
print(type(y))
```

```
<class 'int'>
<class 'int'>
```

In [7]:
```
z = x + y
type(z)
```

Out[7]:
```
int
```

In [8]:
```
z
```

Out[8]:
```
15
```

Multiplying integers together results in an integer.

In [9]:
```
x = 10
y = 5
```

In [10]:
```
z = x * y
type(z)
```

Out[10]:
```
int
```

In [11]:
```
z
```

Out[11]:
```
50
```

Division always results in float

In [12]:
```
x = 10
y = 5
```

In [13]:
```
z = x / y
type(z)
```

Out[13]:
```
float
```

Floats are always displayed with a decimal point even if it is a whole number.

In [14]:
```
z
```

Out[14]:
```
2.0
```

The sum or product of integer with a float results in float

In [15]:
```
x = 10.0
y = 5
print(type(x))
print(type(y))
```

```
<class 'float'>
<class 'int'>
```

In [16]:
```
x + y
```

Out[16]:
```
15.0
```

In [17]:
```
x * y
```

Out[17]:
```
50.0
```

Floating Point Type

A floating point number uses 64 bits to represent decimal values. It can represent many values but only a finite number of distinct values.

A floating point number is capable of approximately 16 places of precision.

It has a maximum value of `1.7976931348623157e+308` which is `sys.float_info.max` (a little less than 2^{1024})

In [18]:
```
2.0 ** 1023
```

Out[18]:
```
8.98846567431158e+307
```

In [19]:
```
2.0 ** 1023 + 2.0 ** 1022 + 2.0 ** 1021
```

Out[19]:
```
1.5729814930045264e+308
```

In [20]:
```
2.0 ** 1024 # this is too big to be represented with 64 bits in double floating point
```

```
---------------------------------------------------------------------------
OverflowError                             Traceback (most recent call last)
Cell In[20], line 1
----> 1 2.0 ** 1024 # this is too big to be represented with 64 bits in double floating point

OverflowError: (34, 'Result too large')
```

Side effect: Floating point numbers do not work the same way real numbers do.

In [21]:
```
a = (1 + 2) / 10
```

In [22]:
```
b = (1/10 + 2/10)
```

In [23]:
```
a == b # with real numbers, we expect these to be equal
```

Out[23]:
```
False
```

In [24]:
```
'{:.20f}'.format(a) # format to print 20 places after decimal
```

Out[24]:
```
'0.29999999999999998890'
```

In [25]:
```
'{:.20f}'.format(b)
```

Out[25]:
```
'0.30000000000000004441'
```

Reference for formatting strings: https://docs.python.org/3/tutorial/inputoutput.html#the-string-format-method

To check if two numbers are approximately equal, you can use `isclose()` in the `math` library.

In [26]:
```
import math
math.isclose(a, b)
```

Out[26]:
```
True
```

Integer type

Integers in Python use variable amounts of memory and can show very large numbers with great precision.

In [27]:
```
2 ** 1023
```

Out[27]:
```
898846567431157953864652595394512366808988489471153286367150405788663379027504815663542386612037680105600569399356967678829394884407208311246423715319737062188883946712432742638151109800623047059726541476042502884419075341171231440736956555270413618581675255342293149119973622969239858152417678164812112068608
```

In [28]:
```
2 ** 1024
```

Out[28]:
```
179769313486231590772930519078902473361797697894230657273430081157732675805500963132708477322407536021120113879871393357658789768814416622492847430639474124377767893424865485276302219601246094119453082952085005768838150682342462881473913110540827237163350510684586298239947245938479716304835356329624224137216
```

In [29]:
```
2 ** 1025
```

Out[29]:
```
359538626972463181545861038157804946723595395788461314546860162315465351611001926265416954644815072042240227759742786715317579537628833244985694861278948248755535786849730970552604439202492188238906165904170011537676301364684925762947826221081654474326701021369172596479894491876959432609670712659248448274432
```

Exponentiation

In [30]:
```
9 ** 2 # power operator. Can result in float or int depending on input.
```

Out[30]:
```
81
```

There is no square root function in base Python

In [31]:
```
sqrt(9)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[31], line 1
----> 1 sqrt(9)

NameError: name 'sqrt' is not defined
```

In [32]:
```
9 ** 0.5 # could work as an alternative to sqrt function.
```

Out[32]:
```
3.0
```

Mathematical constants and many math functions are not defined in base Python. To gain access to common mathematical constants and functions, you must load the `math` library.

In [33]:
```
pi
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[33], line 1
----> 1 pi

NameError: name 'pi' is not defined
```

In [34]:
```
exp(2)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[34], line 1
----> 1 exp(2)

NameError: name 'exp' is not defined
```

In [35]:
```
sin(0)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[35], line 1
----> 1 sin(0)

NameError: name 'sin' is not defined
```

the `math` module

to do math, you must import the `math` module. The `numpy` module will also have a lot of math operations

In [36]:
```
import math
```

In [37]:
```
math.sqrt(9)
```

Out[37]: 3.0

In [38]:
```
math.pi
```

Out[38]: 3.141592653589793

In [39]:
```
math.exp(2)
```

Out[39]: 7.38905609893065

In [40]:
```
math.sin(math.pi / 2) # the math.sin function uses radians
```

Out[40]: 1.0

Boolean Type

Booleans are used to express True or False

In [41]:
```
type(True)
```

Out[41]:
```
bool
```

In [42]:
```
type("True")
```

Out[42]:
```
str
```

There is only one accepted spelling of `True` and `False`. All other spellings will not be the same as the boolean value.

In [43]:
```
type(TRUE) # TRUE or T or t or true
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[43], line 1
----> 1 type(TRUE) # TRUE or T or t or true

NameError: name 'TRUE' is not defined
```

String Type

Strings in Python are created with single or double quotes

In [44]:
```
message1 = "Hello! How are you?"
message2 = 'fine'
```

A few string functions. We'll cover strings more thoroughly in a later lecture

In [45]:
```
len(message1) # number of characters
```

Out[45]: 19

In [46]:
```
4 * message2 # "multiplication" with strings
```

Out[46]: 'finefinefinefine'

Placeholders in strings can be created with curly braces in conjunction with the `.format()` method.

In [47]:
```
name = "Miles"
```

In [48]:
```
"My name is {name} and my school is {school}.".format(name = name, school = "UCLA")
```

Out[48]: 'My name is Miles and my school is UCLA.'

More on the format operator: https://www.w3schools.com/python/ref_string_format.asp

Variables and Assignment

An assignment statement assigns a value to a variable name. It is done with a single equal sign. `=`

The name **must** be on the left-hand side of the equal sign.

The value being assigned must be on the right-hand side of the equal sign.

When an assignment operation takes place, Python will not output anything to the screen.

In [49]:
```
n = 5
```

In [50]:
```
print(n)
```

5

Python Variables are Pointers

Contrast Python to other languages like C or Java. In those languages, when you define a variable, you define a container or 'bucket' that stores a certain kind of data.

```
// C code
int x = 4;
```

The above line defines a 'bucket' in memory intended for integers called `x` and we are placing the value 4 in it.

In Python, when we write

In [51]:
```
x = 4
```

We are defining a *pointer* called `x` that points to a bucket that contains the value 4. With Python, there is no need to "declare" variables.

In Python, we are allowed to have the variable point to a new object of a completely different type. Python is *dynamically-typed*.

We can do the following with no problems:

In [52]:
```
x = 1         # x points to an integer
x = "hello"   # x points to a string
x = [1, 2, 3] # x points to a list
```

Variable Names

You can choose almost anything to be a variable name.

A few rules:

- names can have letters, numbers, and underscore characters `_`
- must not start with a number
- no symbols other than underscore
- no spaces
- cannot be a Python keyword

Python Keywords

```
False      await      else       import     pass
None       break      except     in         raise
True       class      finally    is         return
and        continue   for        lambda     try
as         def        from       nonlocal   while
assert     del        global     not        with
async      elif       if         or         yield
```

The Art of Naming Variables

As you program, do your best to think of good variable names. This is surprisingly hard to do.

The goal is being able to read your program and understand what the variable is without having to go back to the assignment statement to remember.

Some principles (taken from: https://geo-python.github.io/site/notebooks/L1/gcp-1-variable-naming.html)

- Be clear and concise.
- Be written in English.
- Not contain special characters. It is possible to use lämpötila as a varible name, but it is better to stick to ASCII (US keyboard) characters.

Examples of variable names that are not good

In [53]:
```
s = "101533"
```

In [54]:
```
sid = "101533"
```

The above names have the problem that we have no idea what they represent.

In [55]:
```
finnishmeteorologicalinstituteobservationstationidentificationnumber = "101533"
```

This has the problem that it is too long and difficult to read

Examples of variable names that are better

Naming conventions:

- `snake_case` or `pothole_case` uses underscores between words
- `lowerCamelCase` or `UpperCamelCase` uses capital letters to signify new words. lower camel case starts with a lowercase letter, and upper camel case starts with an upper case letter

In [56]:
```
fmi_station_id = "101533"
```

In [57]:
```
fmiStationID = "101533"
```

Other Naming considerations:

Taken from: https://hackernoon.com/the-art-of-naming-variables-52f44de00aad

- It is helpful if the name of a list or array is **plural**.
- If the variable contains string values including `Names` as part of the variable name can be helpful.

In [58]:
```
# not great
fruit = ['apple', 'banana', 'orange']
```

In [59]:
```
# good
fruits = ['apple', 'banana', 'orange']
```

In [60]:
```
# even better as Names implies the usage of strings
fruitNames = ['apple', 'banana', 'orange']
```

Boolean values

Variables containing boolean values are best when they are in the form of a question that can be answered with a yes or no.

In [61]:
```
# not great
selected = True
write = True
fruit = True
```

In [62]:
```
# good
isSelected = True
canWrite = True
hasFruit = True
```

Numeric values

If it makes sense, adding a describing word to the numeric variable can be useful

In [63]:
```
# not great
rows = 3
```

In [64]:
```
# better
minRows = 1
maxRows = 50
totalRows = 3
currentRow = 7
```

Function Names

- functions that modify an object should be named with an **action verb**.
- functions that do not modify an object but return a modified version of the object should be named with a **passive form of a verb**.

For example, a function that will take a list, and modify it by sorting it should be called `sort()`

On the other hand, a function that takes the list, and does not modify the list itself, but simply shows a sorted version of the list can be called `sorted()`

Learn Python by studying Python

The language Python uses many of these best practices for naming functions. You can learn by simply paying attention to how things are written in Python.

In [65]:
```
carBrandNames = ['Ford', 'BMW', 'Volvo', 'Toyota']
carBrandNames.sort() # sorts and modifies the list itself
carBrandNames
```

Out[65]:
```
['BMW', 'Ford', 'Toyota', 'Volvo']
```

In [66]:
```
carBrandNames = ['Chevrolet', 'Audi', 'Honda']
sorted(carBrandNames) # returns the sorted list, but does not modify the list
```

Out[66]:
```
['Audi', 'Chevrolet', 'Honda']
```

In [67]:
```
carBrandNames # we see the list is unmodified
```

Out[67]:
```
['Chevrolet', 'Audi', 'Honda']
```

In [68]:
```
carBrandNames
```

Out[68]:
```
['Chevrolet', 'Audi', 'Honda']
```

In [69]:
```
carBrandNames.sorted() # this attribute does not exist
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[69], line 1
----> 1 carBrandNames.sorted() # this attribute does not exist

AttributeError: 'list' object has no attribute 'sorted'
```

In [70]:
```
sort(carBrandNames) # this function does not exist
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[70], line 1
----> 1 sort(carBrandNames) # this function does not exist

NameError: name 'sort' is not defined
```