Lecture 2 - Programming Concepts

Week 2 Wednesday

Miles Chen, PhD

Adapted from Think Python by Allen B Downey

All Programs can reduced to the following instructions

- *input* - get input from keyboard, a file, network, or some device
- *output* - display data to the screen, save to a file, send over network, etc.
- *math* - perform a mathematical operation
- *conditional execution* - check for certain conditions and run the appropriate code
- *repetition* - perform some action repeatedly, usually with some variation

Values and Types

There are different types of data in Python.

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

Other types include:

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

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

Out[1]:
```
int
```

In [2]:
```
type(2.0)
```

Out[2]:
```
float
```

In [3]:
```
type("2")
```

Out[3]:
```
str
```

In [4]:
```
# sum of integers results in integer
x = 10
y = 5
print(type(x))
print(type(y))
```

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

In [5]:
```
z = x + y
type(z)
```

Out[5]:
```
int
```

In [6]:
```
z
```

Out[6]:
```
15
```

In [7]:
```
# products of integers results in integer
x = 10
y = 5
```

In [8]:
```
z = x * y
type(z)
```

Out[8]:
```
int
```

In [9]:
```
z
```

Out[9]:
```
50
```

In [10]:
```
# division always results in float
x = 10
y = 5
```

In [11]:
```
z = x / y
type(z)
```

Out[11]:
```
float
```

In [12]:
```
z  # floats will always have a decimal point even if it is a whole number
```

Out[12]:
```
2.0
```

In [13]:
```
# sum or product of integer with a float results in float
x = 10.0
y = 5
print(type(x))
print(type(y))
```

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

In [14]:
```
x + y
```

Out[14]:
```
15.0
```

In [15]:
```
x * y
```

Out[15]:
```
50.0
```

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

This number is capable of approximately 16 decimal points of precision.

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

In [16]:
```
2.0 ** 1023
```

Out[16]:
```
8.98846567431158e+307
```

In [17]:
```
2.0 ** 1023 + 2.0 ** 1022 + 2.0 ** 1021
```

Out[17]:
```
1.5729814930045264e+308
```

In [18]:
```
2.0 ** 1024 # this is too big to be represented with 64 bits in double floating point
```

```
---------------------------------------------------------------------------
OverflowError                             Traceback (most recent call last)
<ipython-input-18-2363cf52228b> in <module>
----> 1 2.0 ** 1024 # this is too big to be represented with 64 bits in double floati
ng point

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

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

In [19]:
```
2 ** 1023
```

Out[19]:
```
89884656743115795386465259539451236680898848947115328636715040578866337902750481566354238661203768010560056939935696678829394884407208311246423715319737062188883946712432742638151109800623047059726541476042502884419075341171231440736956555270413618581675255342293149119973622969239858152417678164812112068608
```

In [20]:
```
2 ** 1024
```

Out[20]:
```
17976931348623159077293051907890247336179769789423065727343008115773267580550096313270847732240753602112011387987139335765878976881441662249284743063947412437776789342486548527630221960124609411945308295208500576883815068234246288147391311054082723716335051068458629823994724593847971630483535632962422413721 6
```

In [21]:
```
2 ** 1025
```

Out[21]:
```
35953862697246318154586103815780494672359539578846131454686016231546535161100192626541695464481507204224022775974278671531757953762883324498569486127894824875553578684973097055260443920249218823890616590417001153767630136468492576294782622108165447432670102136917259647989449187695943260967071265924844827443 2
```

In [22]:
```
type(True)
```

Out[22]:
```
bool
```

In [23]:
```
type("True")
```

Out[23]:
```
str
```

In [24]:
```
# There is only one accepted spelling of True and False
# all other spellings will not be the same as the boolean value.
type(TRUE) # TRUE or T or t or true
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-24-0342da79d4e7> in <module>
      1 # There is only one accepted spelling of True and False
      2 # all other spellings will not be the same as the boolean value.
----> 3 type(TRUE) # TRUE or T or t or true

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

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 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

In [25]:
```
7 / 3
```

Out[25]:
```
2.3333333333333335
```

In [26]:
```
7 // 3
```

Out[26]:
```
2
```

In [27]:
```
7 % 3
```

Out[27]:
```
1
```

In [28]:
```
9 ** 2
```

Out[28]:
```
81
```

There is no square root function in base Python

In [29]:
```
sqrt(9)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-29-840f67a85afc> in <module>
----> 1 sqrt(9)

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

In [30]:
```
9 ** 0.5 # could work instead
```

Out[30]:
```
3.0
```

In [31]:
```
pi
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-31-f84ab820532c> in <module>
----> 1 pi

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

In [32]:
```
exp(2)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-32-840a487878a2> in <module>
----> 1 exp(2)

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

In [33]:
```
sin(0)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-33-afbcc558f753> in <module>
----> 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 [34]:
```
import math
```

In [35]:
```
math.sqrt(9)
```

Out[35]:
```
3.0
```

In [36]:
```
math.pi
```

Out[36]:
```
3.141592653589793
```

In [37]:
```
math.exp(2)
```

Out[37]:
```
7.38905609893065
```

In [38]:
```
math.sin(math.pi / 2) # the math.sin function uses radians
```

Out[38]:
```
1.0
```

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 [39]:
```
n = 5
```

In [40]:
```
print(n)
```

```
5
```

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 (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 [41]:
```
s = "101533"
```

In [42]:
```
sid = "101533"
```

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

In [43]:
```
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 [44]:
```
fmi_station_id = "101533"
```

In [45]:
```
fmiStationID = "101533"
```

Other Naming considerations:

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

In [46]:
```
# not great
fruit = ['apple', 'banana', 'orange']
```

In [47]:
```
# good
fruits = ['apple', 'banana', 'orange']
```

In [48]:
```
# 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 [49]:
```
# not great
selected = True
write = True
fruit = True
```

In [50]:
```
# 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 [51]:
```
# not great
rows = 3
```

In [52]:
```
# 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 [53]:
```
carBrandNames = ['Ford', 'BMW', 'Volvo', 'Toyota']
carBrandNames.sort() # sorts and modifies the list itself
carBrandNames
```

Out[53]:
```
['BMW', 'Ford', 'Toyota', 'Volvo']
```

In [54]:
```
carBrandNames = ['Chevrolet', 'Audi', 'Honda']
sorted(carBrandNames) # returns the sorted list, but does not modify the list
```

Out[54]:
```
['Audi', 'Chevrolet', 'Honda']
```

In [55]:
```
carBrandNames # we see the list is unmodified
```

Out[55]:
```
['Chevrolet', 'Audi', 'Honda']
```

In [56]:
```
carBrandNames
```

Out[56]:
```
['Chevrolet', 'Audi', 'Honda']
```

In [57]:
```
carBrandNames.sorted() # this attribute does not exist
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-57-056fced67b59> in <module>
----> 1 carBrandNames.sorted() # this attribute does not exist

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

In [58]:
```
sort(carBrandNames) # this function does not exist
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-58-eea2fd9c8910> in <module>
----> 1 sort(carBrandNames) # this function does not exist

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