Data Science

Introduction to python

CS202 08.31.22

Introduction to Python

Python Data Types

- Text type : str
- Numeric types: int, float, complex
- Sequence types: list, tuple, range
- Mapping type: dict
- Set types: set, frozen set
- Boolean type: bool
- Binary types: bytes, bytearray, memoryview
- None type: NoneType

Intro to Python

Variables

Python, unlike some other languages, does not have a command for declaring variables. A variable is created the moment you assign a value to it. There's also no need to declare a type.

Note: variable names are case sensitive.

```
var_x = “hello world”
```

If you want to declare a data type, this can be done by casting:

```
var_x = str(3)
```
This will give you a string 3 (letter 3)

```
var_x = int(3)
```
This will give you an integer 3

```
var_x = float(3)
```
This will give you a float 3.0

You can always get the data type by:

```
type(var_x)
```

Intro to Python

Variables continued…

We can set many values to multiple variables:

```
x, y, z = “xylophone”, “yolo”, “zebra”
```

Or you can set one value to multiple variables:

```
x = y = z =“yolo”
```

You can always use the print function to print your variables:

```
print(x)
```

This should print “yolo”

Intro to Python

Global variables

Variables that are created outside of a function are called global variables.

```
x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()
```

Variables that are inside a function are local variables and can only be used inside the function.

To create a globale variable inside the function you can use the “global” key word:

```
def myfunc():
  global x
  x = "awesome"

myfunc()

print("Python is " + x)
```

Num Type

Int, float, complex

Int is for integers.

Floats are non integers that includes fraction representation.

Complex are complex numbers.

Example:

```
x = 1      int
y = 3.4   float
z = 2+3j     complex
```

We can always convert using one type to another:

```
convert from int to float:
      a = float(x)

convert from float to int:
      b = int(y)

convert from int to complex:
      c = complex(x)
```

Text type

str

Strings in python are surrounded by quotation marks.

It can be double or single:

`“Hello World”` is the same as `‘Hello World’`

You can display a string literal with the `print( )` function:

```
print(“Hello World”)
```

We can assign a string to a variable:

```
a_str = “Hello World!”
```

You can make a string multiline by using triple quotes:

```
a_para = “”“Hi class!
Nice to meet you all.
My name is Alice. ”””
```

Text type

str

Strings in python are arrays of bytes representing unicode characters.

Square brackets can be used to access elements of the string.

```
a_str = “Hello World!”
print(a[4])
```

Will return you the letter ‘o’

This allows us to perform tasks that we would be able to on arrays such as,

Looping through a string:

```
for letter in “Hello”
  print(letter)
```

Getting length of string:

```
len(“Hello”)
```

Strings

Slicing and Modifying strings

Like arrays you can just return just a range of characters using the slice syntax: the example below should return from position 2 to 5.

```
var_str = "Hello World!"
print(b[2:5]) returns llo
```

If you don’t specify the beginning OR end then they will take the beginning to be 0 and end to be whatever the end is.

We can modify string by using .lower( ) or .upper( ) for lower and upper case respectively:

```
var_str = "Hello World!"
var_str.lower() turns var_str to lower case
var_str.upper() turns var_str to upper case
```

Strings

Slicing and Modifying strings

We can use the strip( ) method to remove white space in the beginning or the end of the string

```
var_str = “ Hello World! “
print(var_str.strip())
```

This will return “Hello World!” Without the white space in the front and back.

You can also replace a string with another string using the replace( ) method:

```
var_str = “Hello World!“
var_str.replace(“H”,”J”)
```

This will return “Jello World!”.

One of the methods we will be using the most is the split( ) method. It will split you string into a list of substrings :

```
var_str = “Hello World!“
 var_str.split(“o”)
```

This will return a list: [“Hell” , “W”, “rld!”]

Strings

Format strings

We can use the format( ) method to include num types into strings:

```
num_apples = 11
txt = “I have {} apples"
print(txt.format(num_apples))
```

This will return you the sentence “I have 11 apples”

Note you can do this many times:

```
num_apples = 11
num_oranges = 12
txt = “I have {} apples and {} oranges“
print(txt.format(num_apples, num_oranges))
```

Boolean Types

Bool

In programming we often want to know if an expression is true or false. For examples if you are splitting a data set of people who’s age is 18 and above for adults and below 18 for children we want to separate using a true or false logic.

i.e. The logic would go as follows: In a data set of 300 people, go through each one, if the person ‘x >= 18’ is true label them adult otherwise if false, label them children.

Here is a much simpler example:

```
a = 20
b = 31

if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

Intro to Python

Operators

- + addition
- - subtraction
- * multiplication
- / division
- % modulus (we can discuss modular arithmetic)
- ** exponentiation
- // floor division (this just gets rid of trailing decimals)

Operators

We have comparison operators:

>= greater than or equal to

<= less than or equal to

> greater than

< less than

!= Not equal to

We can also use words like ‘and’, ‘or’, ‘not’, ‘in’

e.g. x <= 2 and x > 7

e.g. for x in list

We will see more examples of how we can use these operators when we code, these operators will come useful especially when looping through large data sets.

Sequence Types

List, tuple and range

We will first look at lists. We create a list by itemizing inside a [ ] bracket:

```
mylist = ["apple", "banana", "cherry", "orange",
"pear", "cookies!"]
```

Just like an array (or strings!) we can use operations like len( ) and use [ ] to get specific range of items on the list.

```
len(mylist) should return 6
mylist[1] should return “banana”
mylist[2:5] should return [“cherry”, “orange”, “pear”]
```

Sequence types

Lists

We can change an item on the list:

```
fruit_list = ["apple", "banana", "cherry"]
fruit_list[1] = "orange"
```

This will change banana to orange.

We can also change a range of items on the list:

```
fruit_list = ["apple", "banana", "cherry", "orange", "kiwi", “mango"]
fruit_list[1:3] = ["blackcurrant", "watermelon"]
```

This will change banana and cherry to blackcurrant and watermelon respectively.

You can add items by using the .append( ) method or insert an item using the .insert( ) method.

In this course you will find yourself using the .append( ) method the most.

Sequence Types

Methods for lists

We can remove items to a list using the remove( ) method or pop( ) method. The remove( ) method removes the specified item where as the pop( ) method removes by specifying the index of the item you wish to remove.

The sort( ) method will sort the items of your list alphanumerically or numerically in ascending order.

If you want in descending order we will use sort(reverse = True).

copy( ) method allows you to copy a list.

There are many more list methods which I encourage you to look up and try on your own.

Sequence types

List comprehension

Probably the most important thing to know how to do is list comprehension:

List comprehension allows you to write an otherwise long loop into a shorter one:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
  if "a" in x:
    newlist.append(x)

print(newlist)
```

Can be written as:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)
```

Sequence Types

Tuples

Tuples store multiple items in a single variable. Tuples are an ordered collection and they cannot be changed (immutable).

(“Hello”, “world”, “!”) is a tuple.

You can access tuples the same way you access items on a list.

You can add to tuples by changing them to lists or by:

```
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y
```

This should return you the tuple (“apple”, “banana”, “cherry”, “orange”)

You cannot remove items in a tuple.

Tuples

Unpacking

When we create a tuple, we normally assign values to it. This is called "packing" a tuple:

```
fruits = ("apple", "banana", “cherry")
```

We are also allowed to extract the values back into variables. This is called "unpacking":

```
fruits = ("apple", "banana", "cherry")

(green, yellow, red) = fruits

print(green) returns apple
print(yellow) returns banana
print(red) returns cherry
```

If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a list:

```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")

(green, yellow, *red) = fruits

print(green) returns apple
print(yellow) returns banana
print(red) returns [“cherry”, “strawberry”, “raspberry”]
```

Sequence Types

Sets

Sets are made with { } notations, and they are used to store multiple items in a single variable.

We use the add( ) method to add items to a set.

We use the remove( ) method to remove items from a set.

We use the union( ) method to add two sets together unlike in our other sequence types where we can just use ‘+’ method.

We use intersection_update( ) to keep only the items that are in both sets

And symmetric_difference_update( ) to keep only the items that are NOT in both set