python-intro-short

April 3, 2019

1 Python Intro

Felix Biessmann

2 Resources

- Jake Vanderplas: Whirlwind Tour of Python
- Jake Vanderplas: Python Data Science Handbook
- Wes McKinnery: Python for Data Analysis
- Andreas Mueller: Introduction to ML with Python
- Joel Grus: Data Science from Scratch
- Scikit-Learn Documentation
- stackoverflow

3 Why Python?

3.0.1 Popularity

https://stackoverflow.blog/2017/09/06/incredible-growth-python/
https://stackoverflow.blog/2017/09/06/incredible-growth-python/

3.0.2 Libraries

- NumPy
- SciPy
- Matplotlib
- IPython
- Pandas
- sklearn
- tensorflow
- pytorch
- mxnet
- ...

4 How to Install Python

- Most systems come with (an out-dated version of) python pre-installed
- Download binaries for your system at `https://www.python.org/downloads/`
- You can use system specific package managers (apt for linux, homebrew for OSX)
- Or you use **Anaconda** (preferred if you don't like fiddling with installations)

5 Virtual Environments

It is good practice to encapsulate your dependencies in a virtual environment:
Create environment

```
python3 -m venv [path_to_venv]
```

or, if you're using Anaconda

```
conda create -n [name_of_venv]
```

Activate environment:

```
source [path_to_venv]/bin/activate
```

or, if you're using Anaconda

```
conda activate [name_of_venv]
```

6 How to Install Python Packages

In your activated virtual environment you can install packages with
`pip install [packagename]`
We will need jupyter, which brings most other packages we will need:
`pip install jupyter`

7 Starting a notebook

In your activated virtual environment you can now start jupyter with

```
jupyter notebook
```

and a browser window should open.
Click `File` and `New Notebook`

8 The `help` function

In an `ipython` or jupyter shell you can query the documentation of any object with `?`

```
In [8]: b = [1, 2, 3]

In [9]: b?
Type:        list
String Form:[1, 2, 3]
Length:      3
Docstring:
list() -> new empty list
list(iterable) -> new list initialized from iterable's items

In [10]: print?
Docstring:
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file:  a file-like object (stream); defaults to the current sys.stdout.
sep:   string inserted between values, default a space.
end:   string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
Type:      builtin_function_or_method
```

9 Python Syntax

9.1 Intendation!

Whitespace Matters in Python
In Python code is grouped by intendation:

```
for i in [1,2,3,4]:
    if i < 2:
        print("Lower")
    else:
        print("Higher")
```

Don't mix up tabs or spaces!

9.2 Comments Are Marked by

```
# this is a comment and will not be interpreted
```

Python does not have multiline comments

10 Python Variables

- are assigned with =
- are *dynamically typed* (have no static type)
- are pointers

```
In [2]: x = 1          # x is an integer
        print(x)
        x = 'hello'    # now x is a string
        print(x)
        x = [1, 2, 3] # now x is a list
        print(x)
```

```
1
hello
[1, 2, 3]
```

```
In [5]: x = 2 # x is an integer
        x * 2
```

```
Out[5]: 4
```

```
In [6]: x = "2" # x is a string
        x * 2
```

```
Out[6]: '22'
```

11 Some Python Scalar Types

Type	Example	Description
`int`	`x = 1`	integers (i.e., whole numbers)
`float`	`x = 1.0`	floating-point numbers (i.e., real numbers)
`bool`	`x = True`	Boolean: True/False values
`str`	`x = 'abc'`	String: characters or text
`NoneType`	`x = None`	Special object indicating nulls

12 Data Structures

12.1 Builtin Python Data Structures

Type Name	Example	Description
`list`	`[1, 2, 3]`	Ordered collection
`tuple`	`(1, 2, 3)`	Immutable ordered collection
`dict`	`{'a':1, 'b':2, 'c':3}`	Unordered (key,value) mapping

Type Name	Example	Description
`set`	`{1, 2, 3}`	Unordered collection of unique values

12.2 Lists

- Ordered, indexable
- zero-based indexing
- Mutable
- Defined by `[1, 2, 3]`

12.2.1 List Indexing - Accessing Single Elements

```
In [70]: L = [2, 3, 5, 7, 11]
         L[0]

Out[70]: 2

In [71]: L[1]

Out[71]: 3

In [72]: L[-1]

Out[72]: 11

In [73]: L[-2]

Out[73]: 7
```

12.2.2 List Slicing - Accessing Multiple Elements

```
In [74]: L[0:3]

Out[74]: [2, 3, 5]

In [75]: L[:3]

Out[75]: [2, 3, 5]

In [76]: L[-3:]

Out[76]: [5, 7, 11]

In [77]: L[-3:-1]

Out[77]: [5, 7]

In [78]: L[::2]  # equivalent to L[0:len(L):2]

Out[78]: [2, 5, 11]

In [79]: L[::-1] # reverses a list

Out[79]: [11, 7, 5, 3, 2]
```

12.2.3 List Indexing and Slicing for Accessing and Assigning Elements

```
In [80]: L[0] = 100
         L

Out[80]: [100, 3, 5, 7, 11]

In [81]: L[1:3] = [55, 56]
         L

Out[81]: [100, 55, 56, 7, 11]
```

12.3 Lists

Operation	Example	Class
Access	`l[i]`	O(1)
Change Element	`l[i] = 0`	O(1)
Slice	`l[a:b]`	O(b-a)
Extend	`l.extend(...)`	O(len(...))
check ==, !=	`l1 == l2`	O(N)
Insert	`l[a:b] = ...`	O(N)
Delete	`del l[i]`	O(N)
Membership	`x in/not in l`	O(N)
Extreme value	`min(l)/max(l)`	O(N)
Multiply	`k*l`	O(k N)

Source

12.4 Tuples

- Similar to lists
- Immutable
- Defined by `(1, 2, 3)` or `1, 2, 3`

```
In [82]: t = (1, 2, 3)
         t

Out[82]: (1, 2, 3)

In [83]: t = 1, 2, 3
         t

Out[83]: (1, 2, 3)

In [84]: len(t)

Out[84]: 3
```

12.4.1 Elements cannot be changed

```
t[0] = 5
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-86-6dd06f73cec4> in <module>()
----> 1 t[0] = 5

TypeError: 'tuple' object does not support item assignment
```

12.5 Dictionaries

- Hash table
- Extremely flexible and versatile
- Fast access
- Unordered
- Defined by `key:value` pairs within curly braces: `{'a':1, 'b':2, 'c':3}`

```
In [94]: numbers = {'one':1, 'two':2, 'three':3}

In [95]: # Access a value via the key
         numbers['two']

Out[95]: 2

In [96]: # Set a new key:value pair
         numbers['ninety'] = 90
         numbers

Out[96]: {'one': 1, 'two': 2, 'three': 3, 'ninety': 90}
```

12.6 Dictionary

Operation	Example	Class
Access	`d[k]`	O(1)
Change Element	`d[k] = 0`	O(1)
Delete	`del d[k]`	O(1)

Source

13 Operators

- Arithmetic Operators
- Bitwise Operators
- Assignment Operators
- Comparison Operators
- Boolean Operators

- Membership Operators

13.1 Arithmetic Operations

Operator	Name	Description
`a + b`	Addition	Sum of `a` and `b`
`a - b`	Subtraction	Difference of `a` and `b`
`a * b`	Multiplication	Product of `a` and `b`
`a / b`	True division	Quotient of `a` and `b`
`a // b`	Floor division	Quotient of `a` and `b`, removing fractional parts
`a % b`	Modulus	Integer remainder after division of `a` by `b`
`a ** b`	Exponentiation	`a` raised to the power of `b`
`-a`	Negation	The negative of `a`
`+a`	Unary plus	`a` unchanged (rarely used)

```
In [12]: a = 1
         b = 1
         a + b

Out[12]: 2
```

13.2 Assignment Operations

`a += b`	`a -= b`
`a //= b`	`a %= b`
a |= b	`a ^= b`

```
In [2]: a = 2
        a += 2  # equivalent to a = a + 2
        print(a)

4
```

13.3 Boolean Operations

Operator	Description
`a and b`	True if `a` and `b`
`a or b`	True if `a` or `b` is true
`not a`	True if `a` is false.

```
In [7]: x = 4
        (x/2 == 2) and (x * 2 == 8)

Out[7]: True
```

13.4 Comparison Operations

Operation	Description	Operation	Description
`a == b`	a equal to b	`a != b`	a not equal to b
`a < b`	a less than b	`a > b`	a greater than b
`a <= b`	a less than or equal to b	`a >= b`	a greater than or equal to b

```
In [16]: # 25 is odd
         25 % 2 == 1

Out[16]: True

In [17]: # check if a is between 15 and 30
         a = 25
         15 < a < 30

Out[17]: True
```

13.5 Identity and Membership Operators

Operator	Description
`a is b`	True if a and b are identical objects
`a is not b`	True if a and b are not identical objects
`a in b`	True if a is a member of b
`a not in b`	True if a is not a member of b

```
In [18]: 1 in [1,2,3]

Out[18]: True

In [14]: a = [1,2]
         b = [1,2]
         print(f'a is b: {a is b}')
         print(f'a == b: {a == b}')

a is b: False
a == b: True
```

13.6 Strings

- Python is great for Strings

13.6.1 Some Useful String Functions

```
In [35]: message = "The answer is "
         answer = '42'
```

```
In [36]: # length of string
         len(answer)
```

```
Out[36]: 2
```

```
In [37]: # Make upper-case. See also str.lower()
         message.upper()
```

```
Out[37]: 'THE ANSWER IS '
```

```
In [38]: # concatenation
         message + answer
```

```
Out[38]: 'The answer is 42'
```

```
In [39]: # multiplication
         answer * 3
```

```
Out[39]: '424242'
```

```
In [40]: # Accessing individual characters (zero-based indexing)
         message[0]
```

```
Out[40]: 'T'
```

```
In [43]: # finding substrings
         line = 'the quick brown fox jumped over a lazy dog'
         line.find('fox')
```

```
Out[43]: 16
```

```
In [44]: line.find('bear')
```

```
Out[44]: -1
```

```
In [45]: # simple replacements
         line.replace('brown', 'red')
```

```
Out[45]: 'the quick red fox jumped over a lazy dog'
```

```
In [46]: # splitting a sentence into words
         line.split()
```

```
Out[46]: ['the', 'quick', 'brown', 'fox', 'jumped', 'over', 'a', 'lazy', 'dog']
```

```
In [47]: # joining them back together
         '--'.join(line.split())
```

```
Out[47]: 'the--quick--brown--fox--jumped--over--a--lazy--dog'
```

14 Control Flow

- Without control flow, programs are sequences of statements
- With control flow you execute code
- **conditionally** (`if`, `else`)
- **repeatedly** (`for`, `while`)

14.1 Conditional Statements: `if-elif-else`:

```
In [1]: x = inf

        if x == 0:
            print(x, "is zero")
        elif x > 0:
            print(x, "is positive")
        elif x < 0:
            print(x, "is negative")
        else:
            print(x, "is unlike anything I've ever seen...")
```

```
inf is positive
```

14.2 `for` loops

- Iterate over each element of a collection
- Python makes this look like almost natural language:

```
for [each] value in [the] list
```

```
In [2]: for N in [2, 3, 5, 7]:
            print(N, end=' ') # print all on same line
```

```
2 3 5 7
```

```
In [3]: for N in range(5):
            print(N, end=' ') # print all on same line
```

```
0 1 2 3 4
```

14.3 `while` loops

Iterate until a condition is met

```
In [4]: i = 0
        while i < 10:
            print(i, end=' ')
            i += 1
```

```
0 1 2 3 4 5 6 7 8 9
```

15 Functions

Remember the print statement

```
print('abc')
```

print is a function and 'abc' is an argument.

```
In [5]: # multiple input arguments
        print('abc','d','e','f','g')
```

```
abc d e f g
```

```
In [6]: # keyword arguments
        print('abc','d','e','f','g', sep='--')
```

```
abc--d--e--f--g
```

15.1 Defining Functions

```
In [7]: def add(a, b):
            """
            This function adds two numbers

            Input
            a: a number
            b: another number

            Returns sum of a and b
            """
            result = a + b
            return result
```

```
In [8]: add(1,1)
```

```
Out[8]: 2
```

```
In [9]: def add_and_print(a, b, print_result):
            """
            This function adds two numbers

            Input
            a: a number
            b: another number
            print_result: boolean, set to true if you'd like the result printed

            Returns sum of a and b
            """
            result = a + b
            if print_result:
                print("Your result is {}".format(result))
            return result
```

```
In [10]: add_and_print(1, 1, True)

Your result is 2


Out[10]: 2
```

15.2 Default Arguments

```
In [11]: def add_and_print(a, b, print_result=True):
             """
             This function adds two numbers

             Input
             a: a number
             b: another number
             print_result: boolean, set to true if you'd like the result printed

             Returns sum of a and b
             """
             result = a + b
             if print_result:
                 print("Your result is {}".format(result))
             return result
```

```
In [12]: add_and_print(1, 1)

Your result is 2


Out[12]: 2
```

15.3 Anonymous (`lambda`) Functions

```
In [16]: add = lambda x, y: x + y
         add(1, 2)

Out[16]: 3
```

16 Classes

- Python is an object oriented language
- Classes provide a means of bundling data and functionality together
- Classes allow for inheriting functionality

```
In [2]: class Person:

            def __init__(self, name, age):
                self.name = name
                self.age = age

            def is_adult(self):
                return self.age > 18

In [3]: p1 = Person("John", 36)

        print(p1.name)
        print(p1.age)
        print(p1.is_adult())

John
36
True


In [4]: class Student(Person):
            """A class inheriting fields and methods from class Person"""

        p2 = Student("Peter", 20)
        p2.is_adult()

Out[4]: True
```