

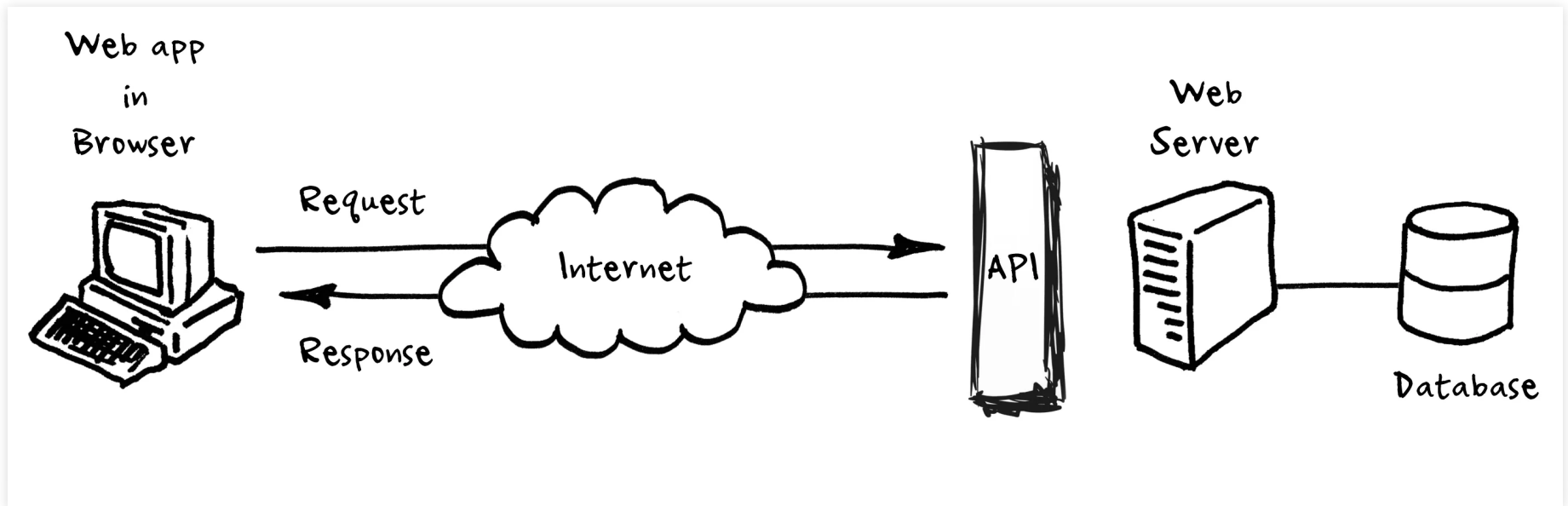
Storytelling with Data

February 5, 2018

Programming Fundamentals

You already got a peek...

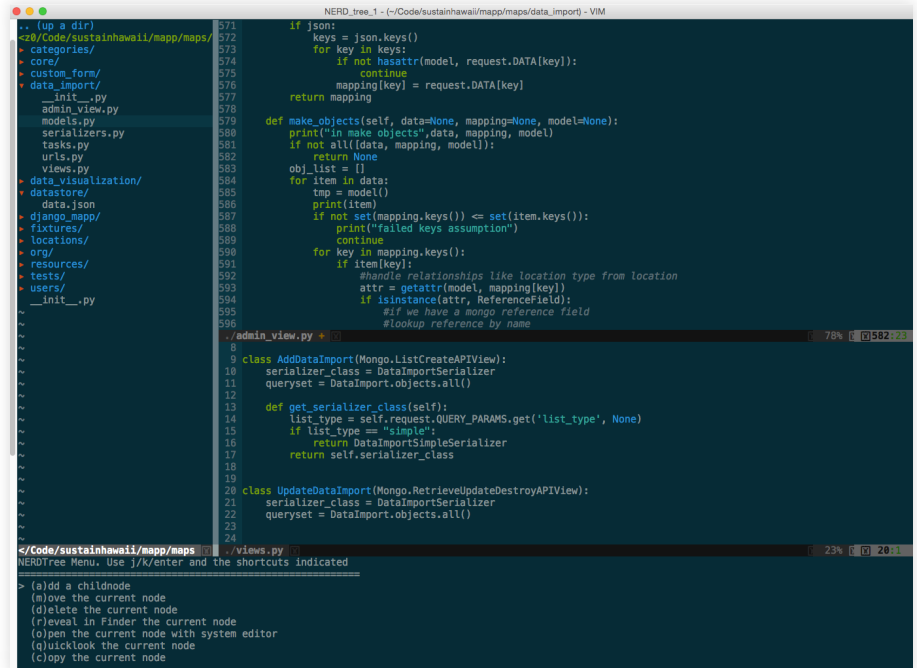
In the second lecture, we showcased a particular task that is well served by use of a programming language: accessing data via an [API](#)



Now let's take some time with the basics.

What is a programming language?

To execute tasks, computers "consume" very low-level binary instructions. It is hard, however, to write instructions as they are consumed because who can keep track of what they mean in a sea of 0s and 1s. Programming languages were developed to provide varying levels of abstractions around certain kinds of low-level tasks. These languages allow us to write prose (of sorts) to communicate our objectives with a computer.



```
.. (up a dir)
~/Code/sustainhawaii/mapp/maps/
├── categories/
├── core/
├── custom_form/
├── data_import/
│   ├── __init__.py
│   ├── admin_view.py
│   ├── models.py
│   ├── serializers.py
│   ├── tasks.py
│   ├── urls.py
│   └── views.py
├── data_visualization/
│   ├── data_store/
│   │   ├── data.json
│   │   ├── django_mapper/
│   │   ├── fixtures/
│   │   ├── locations/
│   │   ├── org/
│   │   ├── resources/
│   │   ├── tests/
│   │   ├── users/
│   │   └── __init__.py
│   └── ...
└── ...
```

```
571 if json:
572     keys = json.keys()
573     for key in keys:
574         if not hasattr(model, request.DATA[key]):
575             continue
576         mapping[key] = request.DATA[key]
577     return mapping
578
579 def make_objects(self, data=None, mapping=None, model=None):
580     print("in make_objects", data, mapping, model)
581     if not all([data, mapping, model]):
582         return None
583     obj_list = []
584     for item in data:
585         tmp = model()
586         print(item)
587         if not set(mapping.keys()) <= set(item.keys()):
588             print("failed keys assumption")
589             continue
590         for key in mapping.keys():
591             if item[key]:
592                 #handle relationships like location type from location
593                 attr = getattr(model, mapping[key])
594                 if isinstance(attr, ReferenceField):
595                     #if we have a mongo reference field
596                     #lookup reference by name
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

```
9 class AddDataImport(Mongo.ListCreateAPIView):
10     serializer_class = DataImportSerializer
11     queryset = DataImport.objects.all()
12
13     def get_serializer_class(self):
14         list_type = self.request.QUERY_PARAMS.get('list_type', None)
15         if list_type == "simple":
16             return DataImportSimpleSerializer
17         return self.serializer_class
18
19
20 class UpdateDataImport(Mongo.RetrieveUpdateDestroyAPIView):
21     serializer_class = DataImportSerializer
22     queryset = DataImport.objects.all()
23
24
```

```
</Code/sustainhawaii/mapp/maps/ views.py 23% [ 20:1
NERDTree Menu. Use j/k/enter and the shortcuts indicated
> (a)dd a childnode
(m)ove the current node
(d)elete the current node
(r)eveal in Finder the current node
(o)pen the current node with system editor
(q)uicklook the current node
(c)opy the current node
```

Programming languages are *easier* than spoken languages

Morphology VERSUS Syntax

Morphology
studies the
structure of words

Syntax studies
the structure of
sentences

Morphemes are
the smallest
units

Words are
the smallest
units

Studies how words
are formed

Studies the word
order and
agreement

Pediaa.com

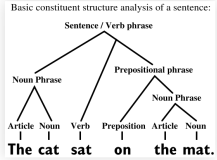
Grammar can be decomposed into **morphology** and **syntax**.

```
public class DurinsBane extends Balrog {  
    private static final int CONTENT_VIEW_ID = 10101010;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        FrameLayout moria = new FrameLayout(this);  
        moria.setId(CONTENT_VIEW_ID);  
        setContentView(moria, new LayoutParams(LayoutParams.MATCH_PARENT, Layout  
  
        if (savedInstanceState == null) {  
            EditText v = new EditText(getActivity());  
            v.setText("Fly you fools!");  
        }  
    }  
}
```

```
public static class Grey extends Gandalf {  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {  
        return inflater.inflate(R.layout.layout_grey, container, false);  
    }  
}
```

YOU SHALL NOT PARSE!

Syntax analysis still works...



Why bother learning to code?























While we will provide you with the option to submit assignments using visualization tools of your choosing, a major goal of this course is to expose you to a programming language. Why?

- Writing code promotes reproducibility because it provides an unambiguous record of decisions
- It promotes iterative development/refinement because duplication costs are low
- Done well, writing code can provide durable tools that can be used for different projects
- Using a general computing language will give you a better tacit sense of how computers operate
- Exposure to programming increases the range of tasks you can accomplish

Why choose Python?

IEEE Spectrum Rating

Take this with a grain of salt...

Language Rank	Types	Spectrum Ranking
1. Python	 	100.0
2. C	  	99.7
3. Java	  	99.5
4. C++	  	97.1
5. C#	  	87.7
6. R		87.7
7. JavaScript	 	85.6
8. PHP		81.2
9. Go	 	75.1
10. Swift	 	73.7

1. Python is very popular
2. There are a ton of **pedagogical tools** available
3. Python helps you do **things other than data analysis**
4. Learning Python **doesn't require a CS background**

What's the **difference** between Python and IPython?

Python is the base language itself. It is what you are actually writing (in most cases), and what is ultimately converted into instructions for the computer.

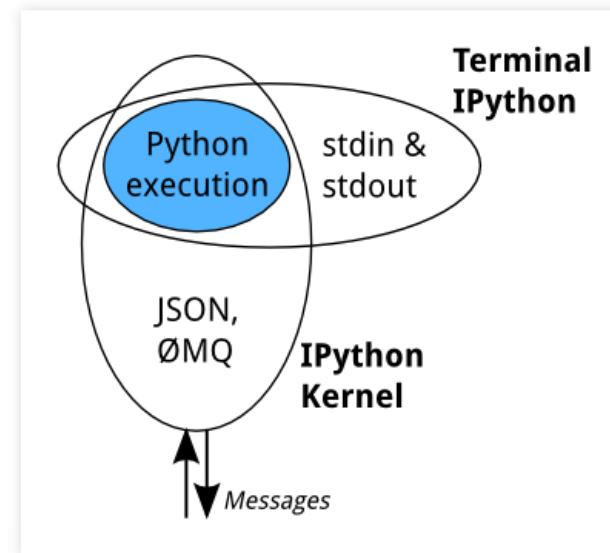
The "I" in **IPython** stands for *interactive*. IPython is an interface that extends the functionality of Python, while also improving the user experience. Among other things, it provides:

- Tab autocompletion
- Clearer and more comprehensive error messages
- Improved command history management
- UNIX shell integration (more on this later)
- A whole mess of graphical user interfaces

Maybe it's just a few GUIs...

... but how does it offer more than one?

The **legit genius** of the IPython project was to separate the **kernel** (the thing that executes code) from the **user interface** (the thing into which you input code).



Why on earth do I care?

That separation makes Jupyter Lab possible...



...and it has implications for your workflow.

The Jupyter project is an offshoot of the IPython project

The [Jupyter Project](#) supports a wide range of languages



On to the live examples...

