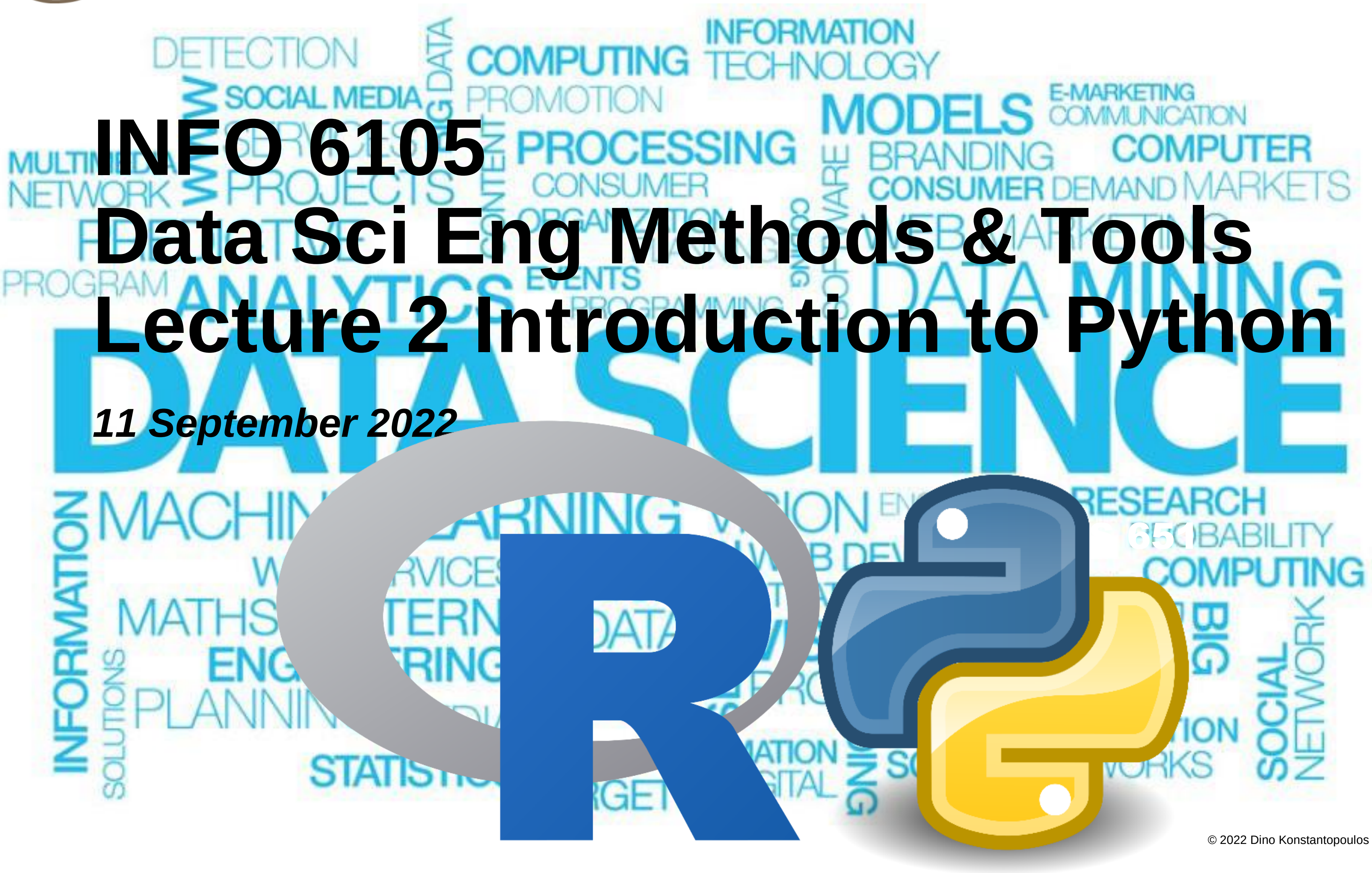
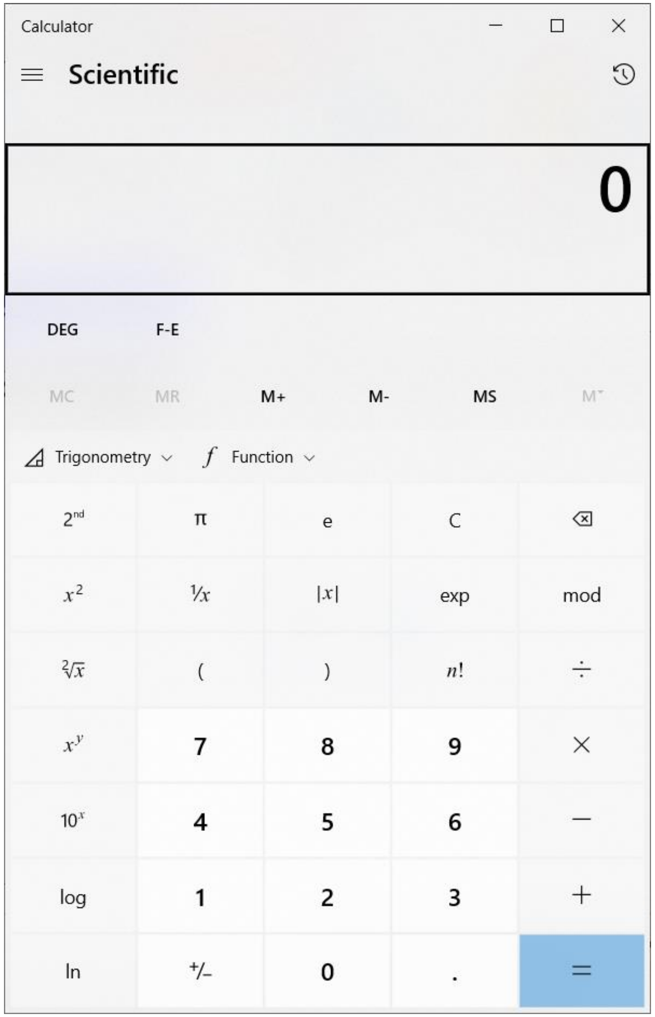
Northeastern University

INFO 6105
Data Sci Eng Methods & Tools
Lecture 2 Introduction to Python

11 September 2022



Programming =



- ..working with many numbers at the same time
- ..storing intermediate computations in variables (likr M+)

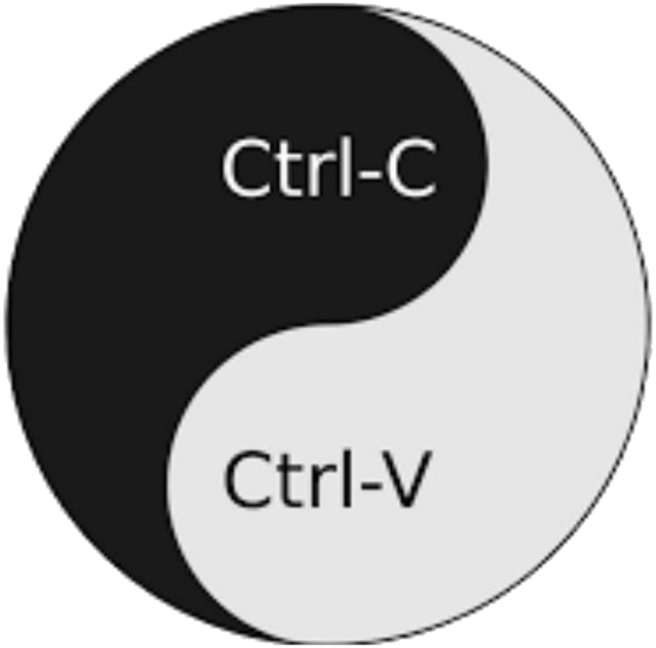
Part 1

PYTHON INTRODUCTION

Python 3.x

- Python is a *managed* language, with its own runtime engine, just like Java and C#. It produces *byte code*!
  - Surprised? Try.. python3 -m compileall .
  - Then try.. import dis; print(dis.dis(myfunction))
  - https://docs.python.org/3/library/dis.html
- On the Mac, it comes installed with, but sometimes it's the *wrong* version for this class
  - You may have python 2.7, but we run python 3.x in class
  - On the Mac, you might have to run python3 instead of python on the command line
  - Make sure to install Anaconda with Python 3.7 (or 3.8 if available)!
- On Windows, you need to install a Python runtime
  - Make sure to install the 3.8 version, not the 2.7 one
  - Make sure to install it on the root of C:\ (not in Program Files)
- Make sure to install 64 bit version of Anaconda, with Python 3.x
  - Run this in a cell: `!python --version`

Python basics



Computing with *many* numbers: Container types

- List: mutable sequence (a "vector")
  - [ ] [23] [23, 45] list('ciao’)
- Tuple: immutable sequence
  - () (23,) (23, 45) tuple('ciao’)
- Set: mutable:
  - {}, set() set((23,)) set('ciao')
  - immutable variant (frozenset)
- *Dict*: map key→value by hashtable
  - {} {2:3} {4:5, 6:7} dict(ci='ao')
- All containers support:
  - `len(c)`, looping (`for x in c`), membership testing (`if x in c`)

Lists

- Most fundamental data structure in Python: An ***ordered*** collection
  - `a = [1,2,3]`
  - `b = [4,5,6]`
  - `List_of_lists = [a, b, []]`
  - `s = sum(a)`
  - `one = a[0]`
  - `three = a[-1]   #last element`
  - `two = a[-2]   #next-to-last eleme`
  - `digits = range(10)`
  - `first_three = digits[:3]`
  - `minus_first_three = digits[3:]`
  - `one_to_four = digits[1:5]`
  - `copy_of_digits = digits[:]`
  - `x.extend([10,11,12])`
  - `x.append(13)`
  - `x, y = [1,2]`

Sorting

-
  ```
  x = [4,1,2,3]
  ```

-
  ```
  y = sorted(x)  #x remains unchanged
  ```

-
  ```
  x.sort()  #x is sorted in place
  ```

-
  ```
  wc = sorted(word_counts.items(),
            key = lambda (word, count): count
            reverse = True) #instead of comparing
            elements themselves, compare results
            of a function that you specify with
            ‘key’
  ```
- Built-in `bisect` module implements binary search and insertion into a sorted list
  - `Bisect.bisect` finds the location where an element should be inserted to keep the list sorted

  -
    ```
    import bisect
    c = [1,2,2,2,3,4,7]
    bisect.bisect(c,2)  #4
    bisect.bisect(c,5)  #6
    bisect.insort(c, 6) #[1,2,2,2,3,4,6,7]
    ```

Tuples

- Lists' ***immutable*** cousins
  - `mylist = [1, 2,3]`
  - `mytuple = (1,2,3)`
- Tuples are a convenient way to return multiple values from functions:

  -
    ```
    def sum_product(x, y):
      return (x + y), (x * y)
    s, p = sum_product(10, 10)
    ```
- Multiple assignments:
  - `x, y = 1, 2`
- Python variable swap:
  - `x,y = y, x`

Dictionaries

- Lists of key/value pairs, or *named arrays*
  - `empty = {}`
  - `also_empty = dict()`
  - `grades = {“dino”: 3.9, “elon”: 4.0}`
  - `elon_grade = grades[“elon”]`
  - `elon_grade = grades.get(“elon”, 0)`
  - `dinograde_p = “dino” in grades`

  -
    ```
    json = {
       “title”: “my blog”,
        “hashtags”: [“#bigdata”, “#crypto”, “#quantum”]
    }
    ```
  - `json.keys()`
  - `json.values()`
  - `json.items()`
- Dictionary keys are immutable

Defaultdict

- Like a regular dictionary, except when you try to look up a key that isn’t there, it first adds a value for it using a zero-argument function you provide when you create it
- Useful whem using dictionaries to collect results by some key and don’t want to check repeatedly for key existence

  -
    ```
    From collections import defaultdict
    word_counts = defaultdict(int)
    for word in document:
      word_counts[word] += 1
    ```

  -
    ```
    dd = defaultdict(dict)
    dd[“dino”][“City”] = “Boston”
                    #{“dino”: {“City”: “Boston”}}
    ```

Counter

```
From collections import Counter
word_count = Counter([“to”, “be”, “or”, “not”,\
“to”, “be”])
```

Sets

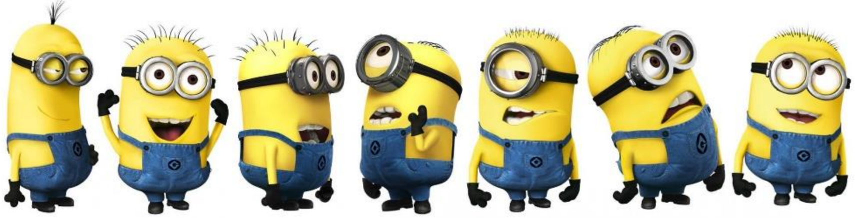
- Unordered collection of ***distinct elements***
  - s = set()
  - s.add(1)
  - s.add(2)
  - s.add(2)
  - p = 2 in s
- **Performance:**
  - stopwords_list = [“a”, “the”, …
    p = “hello” in stopwords #slow
  - stopwords_set = set(stopwords_list)
    p = “hello” in stopw-rds_set #fast
- **Distinct:**
  - word_list = [‘the’, “cat”, “jumps”, …]
  - distinct_word_list = set(world_list)

List comprehensions: Like an R *slice*

- **Remember `matrix(1:16 rows=4, cols-4)` ?**
- **Transformations of lists:**
  - `even_numbers = [x for x in range(100) if x % 2 == 0 ]`
  - `even_set = {x for x in range(100) if x % 2 == 0 }`
  - `zeroes = [ 0 for _ in range(100)]`

  -
    ```
    pairs = [ (x,y)
                   for x in range(100)
                   for y in range(100) ]  #10,000 pairs
    ```

  -
    ```
    some_tuples = [(1,2,3), (4,5,6), (7,8,9)]
    flattened = [x for tup in some_tuples for x in tup]
    flattened  #[1,2,3,4,5,6,7,8,9]
    ```
- **We'll use list comprehensions *a lot* in data science because they represent *anamorphisms* (unfolds or maps) and *catamorphisms* (projections) of data structures**
  - **Get ready for this!**



Generators

- *Generators*, sometimes called *Coroutines*, are sequences you can iterate over, but which are only produced ***lazily (as needed)***
- You can create generators with functions and yield:

  -
    ```
    def lazy_range(n):
      """a lazy version of range()"""
      i = 0;
      while i < n:
        yield i
        i += 1
    ```

  -
    ```
    # to consume yielded values:
    for i in lazy_range(10)
      print(i)
    ```

- You may also create generators by using list comprehensions wrapped in parenses:
  - `lazy_ints_under_100 = (i for i in range(100))`

Iterators

- An *eager* structure (opposite of lazy)
- Standard `itertools` library has a collection of generators for common data algorithms

```
import itertools
first_letter = lambda x: x[0]
names = [ ‘Alex’, ‘Aria’, ‘Wally’, ‘Will’,
‘Ariana’, ‘Steve’]
for letter, names in itertools.groupby(names,
first_letter):
  print(letter, list(names))
 # A [‘Alex, ‘Aria’]
   W [‘Wally’, ‘Will’]
   A [‘Ariana’]
   S [‘Steve’]
```

Control flow

-
  ```
  if 1 == 2:
    print(“uh-oh”)
  elif 1 == 3:
    print(“uh-oh-again”)
  else:
    print(“whew..”)
  ```

-
  ```
  x = 0
  while x < 100:
    print(x)
    x += 1 #x = x + 1
  ```

-
  ```
  for x in range(100):
    if x < 100:
      continue
    if x > 100:
      break;
    print(x)
  ```

Enumerations

```
#nicely functional
for (i, document) in enumerate(documents):
  do_something(i, document)
```

```
#unpythonic
for i in range(len(documents)):
  document = documents[i]
  do_something(I, document)
```

```
#also unpythonic
i = 0
for document in documents:
  do_something(i, document)
  i += 1
```

File IO

-
  ```
  path = ‘myfolder/mybigdata.txt’
  ```

-
  ```
  f = open(path)
  for line in f:
    print(line)
  #EOL marker intact
  ```

-
  ```
  Lines = [x.rstrip() for x in open(path)]
  #EOL-free
  ```
- **Using analog:**

  -
    ```
    with open(path) as f:
      lines = [x.rstrip() for x in f]
    #automatically closes the file when exiting
    with block
    ```

-
  ```
  with open(path, ‘rb’) as f:
    data.decode(‘utf8’)
  ```

Object Oriented Python

```
Class Set:
  def __init__(self, values=None):
    “””ctor”””
    self.dict = {} #each instance has its own
                   #dict which is what we use
                   #to track membership
    if values is not None:
      for value in values:
        self.add(value)

  def add(self, value):
    self.dict[value] = True

  def contains(self, value):
    return value in self.dict

  def remove(self, value):
    del self.dict[value]
```

Currying

- **Partially applying functions to create new functions**

  - def exp(base, power):
    return base ** power
  - def two_to_the(power):
    return exp(2, power)
  - From functools import partial
    two_to_the = partial(exp, 2)
  - Print(two_to_the(3))

Function Oriented puzzle

- Let’s say we want to create a higher-order function that takes as input some function `f` and returns a new function that for any input returns twice the value of `f`

```
def dpubler(f):
  def g(x):
    return 2 * f(x)
  return g
```

- Works in most cases:

```
def f_plus_1(x)::
  return x + 1;
g = doubler(f_plus_1)
print(g(3))  # 8 = (3 + 1) * 2
```

- But:

```
def sum(x, y)::
  return x + y;
g = doubler(sum)
print(g(1,2))
```

args and kwargs

- **What we need is a way to specify a function that takes arbitrary arguments:**

```
def magic(*args, **kwargs):
  print (“unnamed args: “, args)
  print (“keyword args: “, kwargs)
magic(1,2, key1 = “nu”, key2 = ’rocks!”);
```

- **`args` is a tuple of its unnamed arguments and `kwargs` is a dictionary of its named arguments. So now we can:**

```
def dpublerr(f):
  “””works no matter the inputs”””
  def g(*args, **kwargs):
    “””pass all arguments to f”””
    return 2 * f(*args, **kwargs)
  return g
```

- **And now:**

```
g = doublerr(sum)
print g(1, 2)  # 6:
```

Zippers

- zip transforms multiple lists into a single list of tuples of corresponding elements
  - list1 = [‘a’, ‘b’, ‘c’]
  - list2 = [1, 2, 3]
  - zipper = zip(list1, list2) #[(‘a’, 1), (‘b’, 2), (‘c’, 3)]
  - Orig_letters, orig_numbers = zip(*zipper)
             # * performs argument unpacking
  - def add(a, b): return a + b
  - add(1, 2)   #3
  - add([1,2]) #TypeError!
    add(*[1,2]) #3

practice your Python

- To practice:
  - Find python videos on youtube or good MOOCs
  - The good ones are those that don't put you to sleep after 10 minutes. If you're still awake after 10 minutes and you feel like you're learning, then..
- Examples:
  - https://www.udemy.com/course/python-exercises/ ($16)
  - https://www.udemy.com/course/automate ($16)
  - https://www.coursera.org/learn/python-crash-course (free)
  - https://www.youtube.com/user/khanacademy/search

CLASS DISMISSED
iStockphoto