# Introduction to Data Science – Week 3

NETA LIVNEH

COURSE 55793

2019-2020

# Last Week Recap

Data types:

- Lists
- Tuples
- Booleans

Iterators

Conditional statements (ifs)

Loop statements:

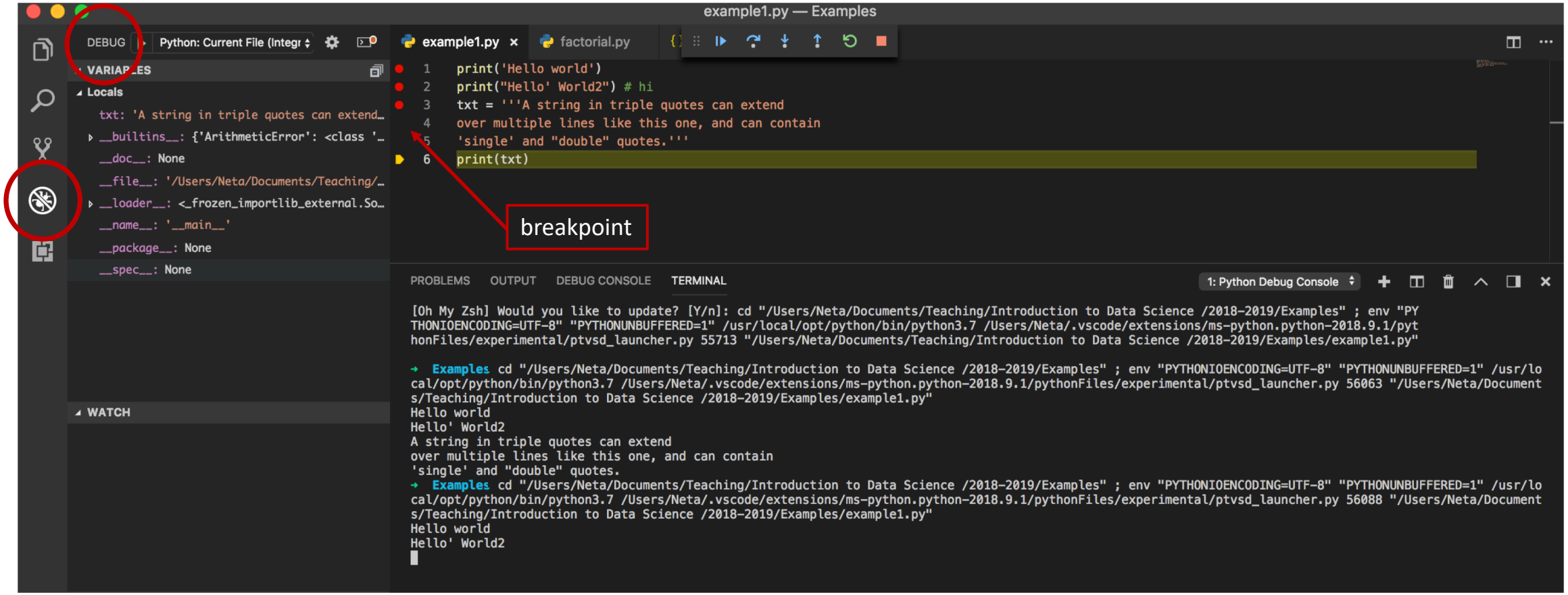
- While

Debugging

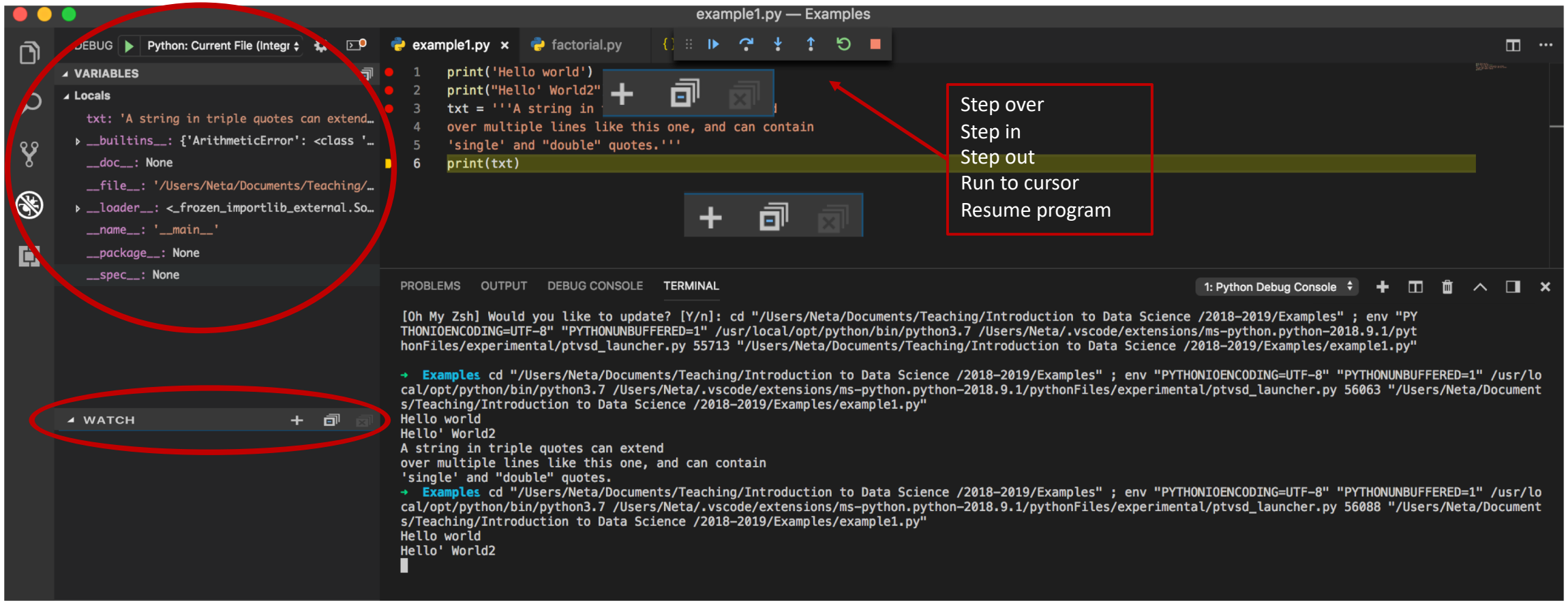
We encountered these built-in functions: input(), int(), list.append(x), range()



# Debugging - breakpoints



# Debugging – Steps



# This Week

For loop statements

Functions

Comments

Data types: Sets



# For loop

For statement iterates over the items of any ***sequence*** (we have already met list, string, tuple, but this is also true for dict and iterators) in the order that they appear in the sequence.

Exempels:

```
In [89]: for day in ('Sun', 'Non', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat'):
    ...:     print(day)
    ...:
Sun
Non
Tue
Wed
Thu
Fri
Sat

In [90]: s = 0
In [91]: for i in range(1,10):
    ...:     s += i
    ...:
In [92]: print(s)
45
```

```
for <variable> in <sequence>:
    <statements>
else: <statements>
```



# Nested For

Nested loop is executed for each iteration of the outer for:

```
>>> combs = []
>>> for x in [1,2,3]:
...     for y in [3,1,4]:
...         if x != y:
...             combs.append((x, y))
...
>>> combs
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```



# Iterators

An object is considered ***iterable*** if it is either a physically stored sequence (e.g. list, tuple, string, file), or an object that ***produces one result at a time*** in the context of an iteration tool like a for loop.

The main advantage is that we don't need to create the sequence in advance or store all of it in memory.

Very useful for reading files line by line.



# Functions – built-in functions

A function is a structuring element in programming languages to group a set of statements so they can be utilized more than once in a program.

Functions offer a way to

- reuse and share code
- help organize the program

Functions can receive arguments and return values

Python has few built-in functions. Other functions reside in libraries and can be added manually.

For example:

```
int("40") # Converts string to integer number
str(40) # Converts number to string
```



# Functions

```
def functionname(parameters):
    "function_docstring"
    function_suite
    return (expression)
```

Notes:

- begin with the keyword *def* followed by the *function name,* parentheses ( ( ) ) and ends with a colon (:)
- Input parameters or arguments should be placed within these parentheses.
- The first statement of a function (optional) is the documentation string of the function or *docstring*.
- The code block within every function is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller.
- When a function doesn't return anything or when return is missing, it actually return None.



# Functions - Example

```
In [94]: def factorial(num=1):
    ...:     """
    ...:     Calcualtes the factorial of the integer number in num. if num is not integer,
    ...:     return None.
    ...:     Returns: num!
    ...:     """
    ...:     if type(num) is not int:
    ...:         print('{} in not an integer. Please entel a positive integer number'.format(num))
    ...:         retrun
    ...:     if num < 0:
    ...:         print('{} in negative. Please enter a positive integer number'.format(num))
    ...:         return
    ...:     total = 1
    ...:     for i in range(1, num + 1):
    ...:         total *= i
    ...:     return(total)
    ...:

In [95]: factorial(5)
Out[95]: 120

In [96]: def factorial2(num=1):
    ...:     total = 1
    ...:     if num > 1:
    ...:         total = num * factorial2(num-1)
    ...:     return(total)
    ...:
```



# Functions: Default Argument Values

```
In [97]: def ask_ok(prompt, retries=4, reminder='Please try again!'):
     ...:     while True:
     ...:         ok = input(prompt)
     ...:         if ok in ('y', 'ye', 'yes'):
     ...:             return True
     ...:         if ok in ('n', 'no', 'nop', 'nope'):
     ...:             return False
     ...:         retries = retries - 1
     ...:         if retries < 0:
     ...:             raise ValueError('invalid user response')
     ...:         print(reminder)
```

This function can be called in several ways:

giving only the mandatory argument: `ask_ok('Do you really want to quit?')`

giving one of the optional arguments: `ask_ok('OK to overwrite the file?', 2)`

or even giving all arguments: `ask_ok('OK to overwrite the file?', 2, 'Come on, only yes or no!')`



# Functions and SCOPE of variables

What is a scope of a variable?

Why scope is important?

Local variables:

- Variables assigned in functions
- Function arguments

Change of local variables won't effect external

```
pi = 3.14
def area(r):
    return pi * r * r
```

```
x = 0
y = 0
def incr(x):
    y = x + 1
    return y
incr(5)
print x, y
```

Global variables

Local variables

Global variables

```
numcalls = 0
def square(x):
    global numcalls
    numcalls = numcalls + 1
    return x * x
```

**global keyword**

Use within functions to create globally accessible variables



# Data Types: Sets

A set is similar to a list but it cannot have multiple occurrences of the same element.

A set contains an *unordered* collection of *unique* and *immutable* objects (can't have lists).

Example:

```
In [115]: {1, 2, 3, 3, 4}
Out[115]: {1, 2, 3, 4}
In [116]: type(_)
Out[116]: set


In [117]: cities = set(("Paris", "Lyon", "London","Berlin","Paris","Birmingham"))
In [118]: cities
Out[118]: {'Berlin', 'Birmingham', 'London', 'Lyon', 'Paris'}
In [119]: empty_set = set()
```



# Operations on Sets

| Operation | Equivalent | Result |
|---|---|---|
| s.add(x) | | Adds an element x, which has to be immutable, to a set s if it is not already there. |
| s.pop() | | removes and returns an arbitrary set element. The method raises a KeyError if the set is empty. |
| s.discard(x) | | Removed x from set s. If x is not in s there is no change. |
| s.remove(x) | | Removed x from set s. The method raises a KeyError if x s not in set s. |
| s1. difference(s2) | s1-s2 | Returns the difference of two or more sets as a new set |
| s1. difference_update(s2) | s1 = s1 - s2 | Removes all elements of another set s2 from set s1. |
| s1.union(s2) | s1 \| s2 | Returns the union of two sets as a new set, i.e. all elements that are in either set s1 or s2. |
| s1.intersection(s2) | s1 & s2 | Returns the intersection of set s1 and set s2 as a new set. |
| s1.isdisjoint(s2) | | returns True if two sets have a null intersection |
| s1.issubset(s2) | s1 <= s2; s1 < s2 | Returns True if s2 is a subset of s1. < for proper subset. |
| s1.superset(s2) | s1 >= s2; s1 > s2 | Returns True if s2 is a superset of s1. < for proper superset. |



# Operations on Sets

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> fruit = set(basket)               # create a set without duplicates
>>> fruit
set(['orange', 'pear', 'apple', 'banana'])
>>> 'orange' in fruit                 # fast membership testing
True
>>> 'crabgrass' in fruit
False

>>> # Demonstrate set operations on unique letters from two words
...
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                                 # unique letters in a
set(['a', 'r', 'b', 'c', 'd'])
>>> a - b                             # letters in a but not in b
set(['r', 'd', 'b'])
>>> a | b                             # letters in either a or b
set(['a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'])
>>> a & b                             # letters in both a and b
set(['a', 'c'])
>>> a ^ b                             # letters in a or b but not both
set(['r', 'd', 'b', 'm', 'z', 'l'])
```



# Class exercise: Fibonacci

Write a program that asks the user how many Fibonnaci numbers to generate and then generates them. The function returns the sequence.

- The Fibonnaci sequence is a sequence of numbers where the next number in the sequence is the sum of the previous two numbers in the sequence. The sequence looks like this: 1, 1, 2, 3, 5, 8, 13, …)
- Don’t forget to initialize the list

# Fibonacci

```
>>> def fib2(n):  # return Fibonacci series up to n
...     """Return a list containing the Fibonacci series up to n."""
...     result = []
...     a, b = 0, 1
...     while a < n:
...         result.append(a)    # see below
...         a, b = b, a+b
...     return result
...
>>> f100 = fib2(100)    # call it
>>> f100                # write the result
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```