Introduction to Data Science – Week 2

NETA LIVNEH

COURSE 55793

2019-2020

Last Week Recap

IDE – for working with scripts and files

Playground/Terminal/Command Line/Prompt

Numbers: Integers and floats

What is a Variable

Strings

Indexing and Slicing

We encountered these functions: print(), type()

This Week

Data types:

- Lists
- Tuples
- Booleans

Iterators

Conditional Statements (Ifs)

Loop Statements:

- While
- For

Debugging

Data Types - Lists

They are ordered

The contain arbitrary objects

Elements of a list can be accessed by an index

They are arbitrarily nestable, i.e. they can contain other lists as sublists

Variable size

They are mutable, i.e. the elements of a list can be changed

```
In [41]: li0 = []

In [42]: len(li0)
Out[42]: 0
```

Empty list

```
In [43]: li = [1, 2, 3, 4, 5]

In [44]: li = li + [6]
In [45]: li.append(7)
In [46]: li
Out[46]: [1, 2, 3, 4, 5, 6, 7]
```

Slicing and Indexing (zero based):

```
In [47]: li[0] = 'this is a list'

In [48]: li[2:5]
Out[48]: [3, 4, 5]

In [49]: li[::2]
Out[49]: ['this is a list', 3, 5, 7]

In [50]: li2 = [8,9]

In [51]: li[-3:-1] = li2

In [52]: li
Out[52]: ['this is a list', 2, 3, 4, 8, 9, 7]
```


Operations on Lists

Method	Description	Method	Description
list.append(x)	add item to the end. Same as a[len(a):] = [x]	list.extend (list2)	Appends the contents of list to list
list.insert(index, x)	Inserts object xinto list at offset index	list.remove(x)	Remove the first item from the list whose value is x. It is an error if there is no such item.
list.pop()	removes and returns the last item in the list	list.index(x)	Return the index in the list of the first item whose value is x. It is an error if there is no such item.
list.count(x)	Return the number of times x appears in the list.	list.sort([func])	Sorts objects of list, use compare func if given
list.reverse()	Reverse the elements of the list, in place.	del list[i]	Delete value at index I from list

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.

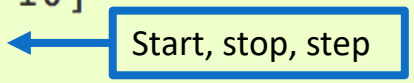
Very useful for reading files line by line.

Using range()

https://docs.python.org/3/library/stdtypes.html#typesseq-range

Range examples:

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(1, 11))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list(range(0, 30, 5))
[0, 5, 10, 15, 20, 25]
>>> list(range(0, 10, 3))
[0, 3, 6, 9]
>>> list(range(0, -10, -1))
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
>>> list(range(0))
[]
>>> list(range(1, 0))
[]
```



Data Types - Tuples

A tuple is a sequence of immutable Python objects (similarly to lists).

Tuples use parentheses, whereas lists use square brackets.

Tuples are immutable.

Example:

```
In [63]: tup1 = 'Math', 'Statistics', 'Programming', 2017

In [64]: tup2 = (1, 2, 3)

In [65]: print(tup1 + tup2)
('Math', 'Statistics', 'Programming', 2017, 1, 2, 3)

In [66]: tup3 = (tup1, tup2)
```

Packing

```
In [67]: print(tup3)
(('Math', 'Statistics', 'Programming', 2017), (1, 2, 3))

In [68]: print(tup3[0][2])
Programming
```


Data Types - Tuples

```
In [69]: tup1[3] = 2018
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-70-03d2b292dbf0> in <module>()
----> 1 tup1[3] = 2018

TypeError: 'tuple' object does not support item assignment

In [70]: tup1, tup2 = tup2, tup1
```

Swizzling

```
In [71]: print(tup1, tup2, tup3)
(1, 2, 3) ('Math', 'Statistics', 'Programming', 2017) (('Math', 'Statistics', 'Programming', 2017), (1, 2, 3))

In [70]: tup4 = 'tuple of size 1',

In [74]: (a, b, c) = tup1
```

Unpacking

```
In [75]: print('a:', a, 'b:', b, 'c:', c)
a: 1 b: 2 c: 3
```


Variables Types: Booleans

Takes values of either **True** or **False**.

For numbers, any non-zero value is true; zero is false.

More generally, anything with a non-zero length is true; empty sequences or *none* responses are false.

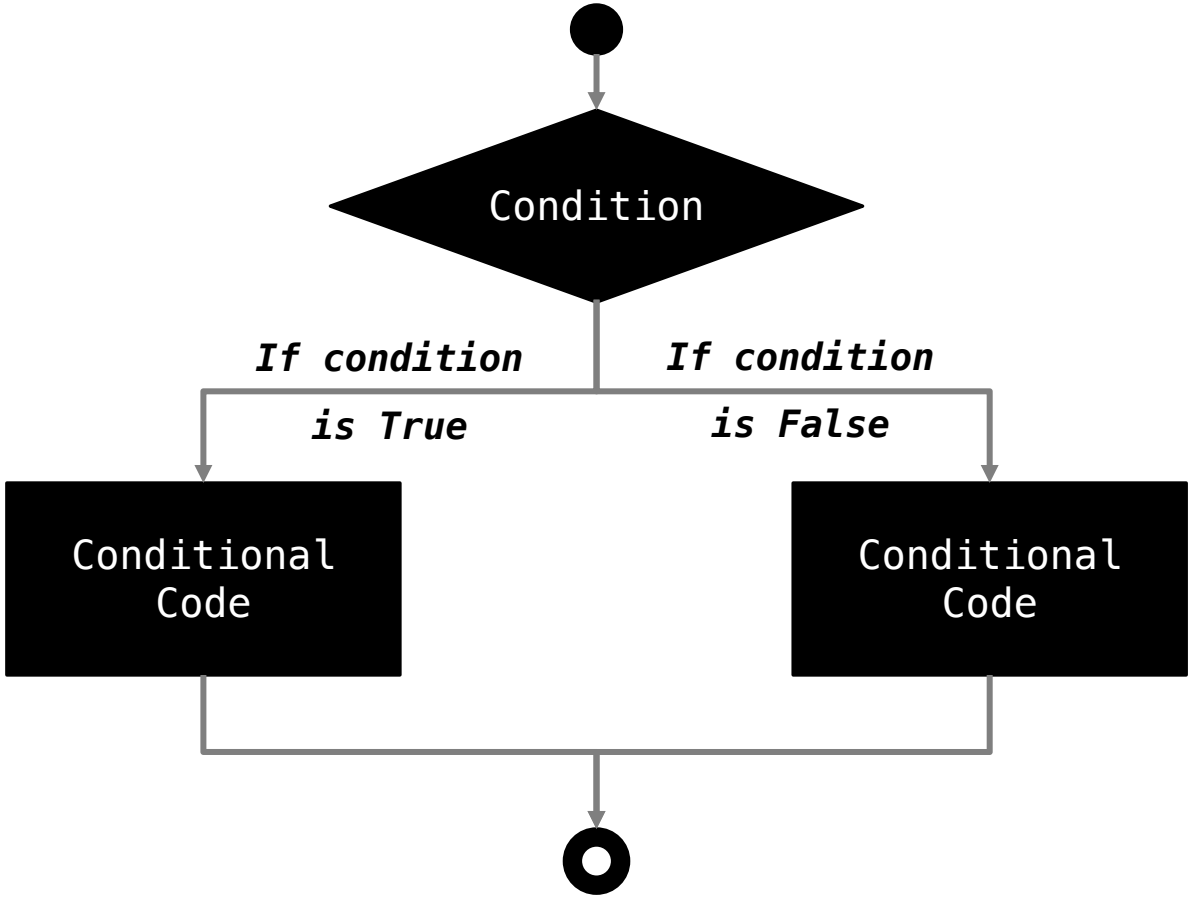
Sign	Meaning
a == b	The value of a is equal to the value of b.
!=	Not equal
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal
a is b	a and b point to the same value in memory.

And	True	False
True	True	False
False	False	False

Or	True	False
True	True	True
False	True	False

Not	True	False
	False	True

Conditional Statements (Ifs)




```
In [76]: x = 12

In [77]: if x > 10:
    ...:     y = 14
    ...: else:
    ...:     y = 7
    ...: print(y)
    ...:
14
```

Indentation

Conditional Statements (Ifs)

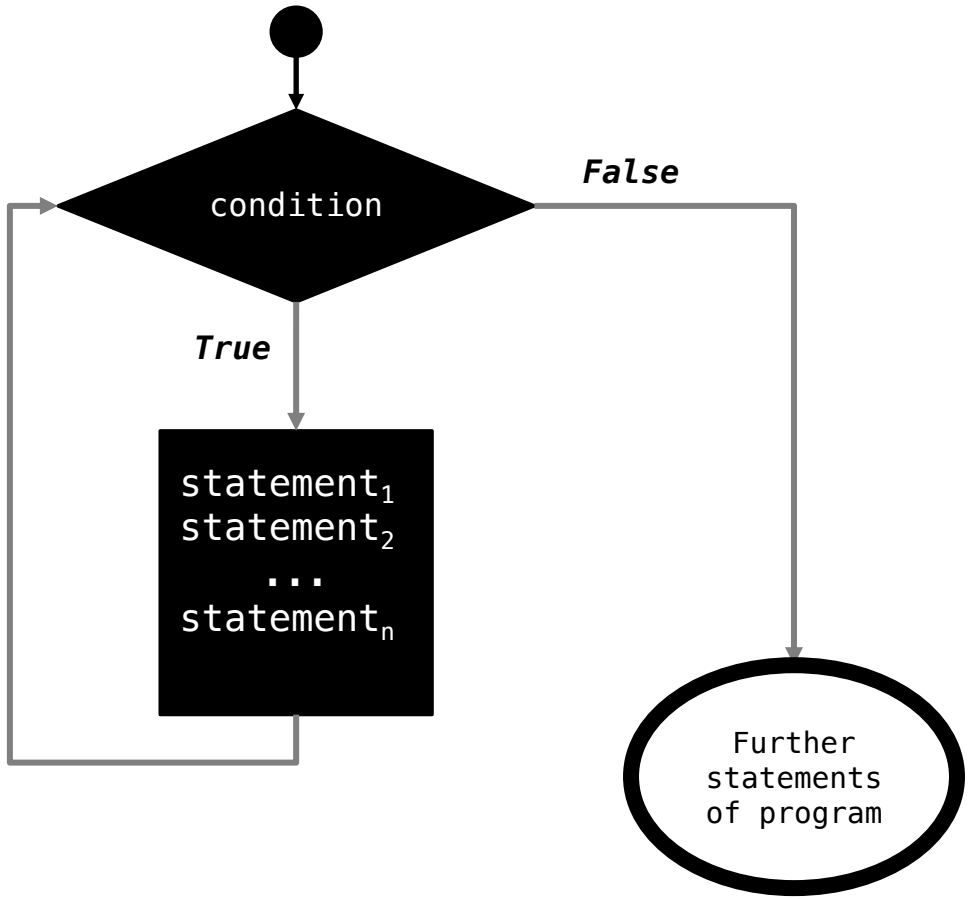
```
In [81]: tired = int(input('How tired are you or a scale of 1 to 10: '))
# eval(input()) would also work here
How tired are you or a scale of 1 to 10: 8


In [82]: if tired <5:
    ...:     print('Great! then lets continue')
    ...: elif tired < 8:
    ...:     print('Lets take a break')
    ...: else:
    ...:     print('OK, lets stop for today')
    ...:
OK, lets stop for today
```


While Loop

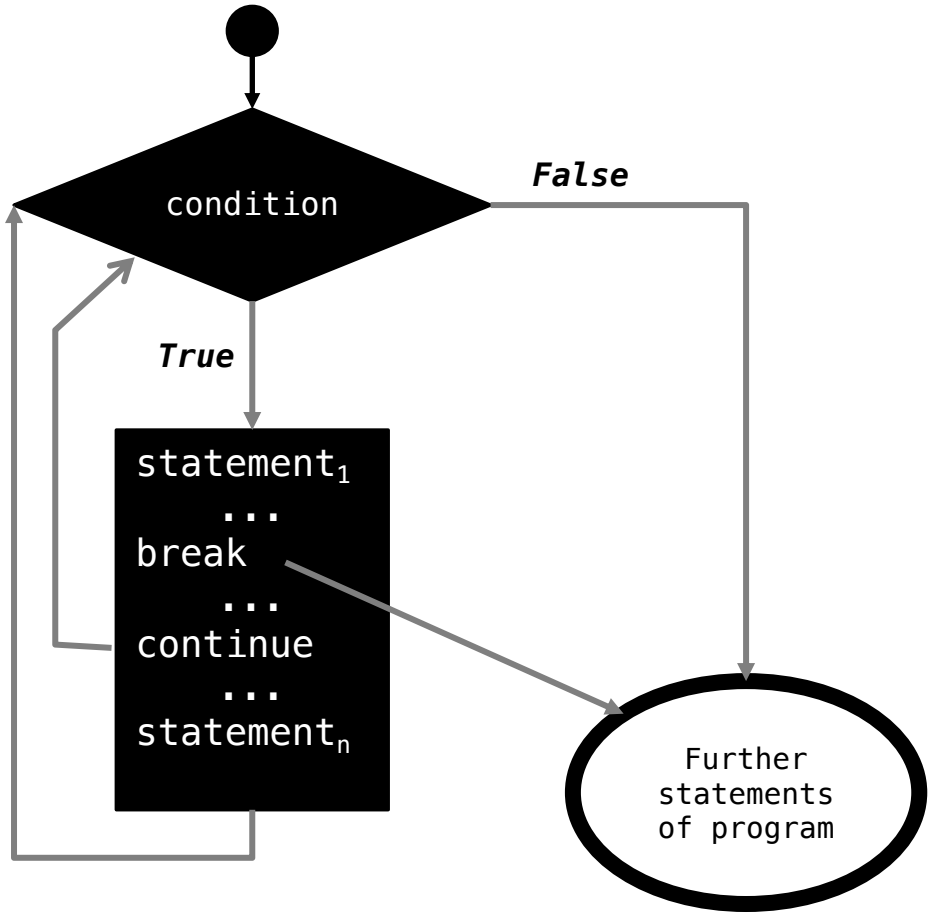
Loops make it possible to carry out a sequence of statements repeatedly. The code within the loop, i.e. the code carried out repeatedly, is called the body of the loop

The while loop executes as long as the condition is true




```
While <condition>:
    <statements>
else: <statements>
```

While Loop




```
In [87]: count = 0

In [88]: while count < 5:
    ...:     print('You have {} more tries'.format(5-count))
    ...:     count += 1
    ...:
    ...: print('All done')
    ...:
You have 5 more tries
You have 4 more tries
You have 3 more tries
You have 2 more tries
You have 1 more tries
All done
```

Loop control statements

Control Statement	Description
break	Terminates the loop statement and transfers execution to the statement immediately following the loop
continue	Causes the loop to skip the remainder of its body and immediately retest its condition (or sequence) prior t reiterating
pass	*Null* operation; nothing happens when it executes. The pass is also useful in places where your code will eventually go, but has not been written yet

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>
```


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)]
```


Debugging

How to find bugs in your script:

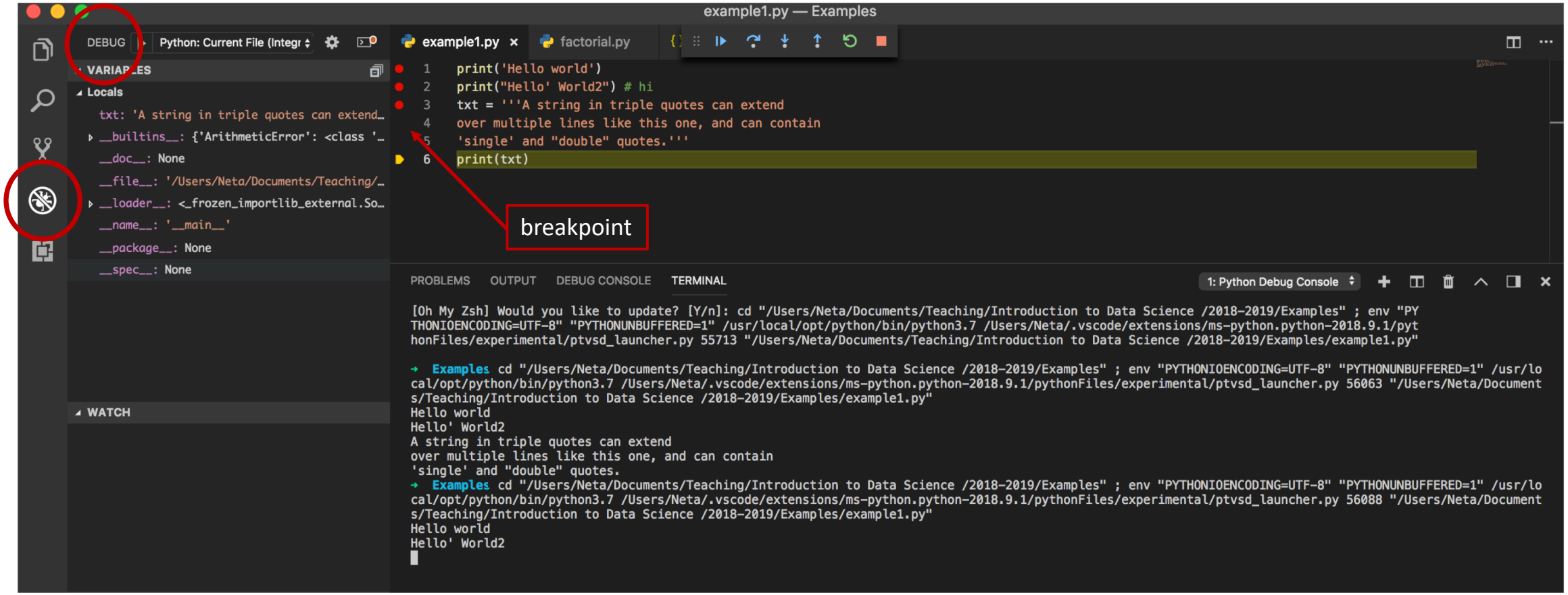
Use a lot of print statements to understand what is going on

Use *debug mode* and *breakpoints* to stop your script at some line, and run your script line by line (using F10). Follow the variables change in the debugger.

Comment out (i.e., #) statements to check if they are the problem or if you don't want to run them.

- To comment out a block use Shift + /

Debugging - breakpoints



Debugging – Steps