basics-part1-presentation

October 27, 2015

1 Python basics - part I

1.1 Variables, assignments & data types

```
In [15]: #name = object
         a = 17
         b = "bla"
```

```
In [19]: a
```

```
Out[19]: 17
```

```
In [20]: b
```

```
Out[20]: 'bla'
```

```
In [21]: print(a)
         print(b)
```

```
17
bla
```

```
In [22]: print(c)
```

```
        ---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call last)

        <ipython-input-22-5315f3e3adca> in <module>()
    ----> 1 print(c)


        NameError: name 'c' is not defined
```

```
In [7]: a = 17
        a = "Used names can always be reassigned to other objects regardles of their data type!"
        print(a)
```

```
Used names can always be reassigned to other objects regardles of their data type!
```

```
In [9]: speak = print
        speak("even functions are objects and can be assigned to variables")
```

```
even functions are objects and can be assigned to variables
```

```
In [25]: print(type("Some bacis data types:"))
         print(type(3))
         print(type(3.14))
         print(type(True))
```

```
<class ’str’>
<class ’int’>
<class ’float’>
<class ’bool’>
```

Conversion of datatypes:

```
In [31]: print(int(5.5))
         print(float(’5.23’))
         print(str(12))
         print(bool(’True’))
```

```
5
5.23
12
True
```

1.2 Arithmetic

- –
- –
- –
- /
- // (integer division)
- % (modulo operator)
- ** (power)

```
In [29]: a = 3
         b = 3.5
```

```
In [30]: a+b
```

```
Out[30]: 6.5
```

```
In [32]: 3*a
```

```
Out[32]: 9
```

```
In [33]: c = "bla"
         d = "blub"
```

```
In [34]: c + d
```

```
Out[34]: ’blablub’
```

```
In [1]: #sometimes its possible to mix data types
        2 * "hey " + "wicky"
```

```
Out[1]: ’hey hey wicky’
```

```
In [45]: #but not everything makes sense
         12 + "monkeys"


        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-45-0c5049cae2fe> in <module>()
          1 #but not everything makes sense
    ----> 2 12 + "monkeys"


        TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
In [22]: 2**10

Out[22]: 1024
```

1.3 Conditions and control statements (if, while)

1.3.1 Comparison operators:

Operator	True, if
a == b	a equals b
a > b	a is larger than b
a < b	a is smaller than b
a >= b	a is larger than b or equals b
a <= b	a is smaller than b or equals b
a != b	a and b are unequal
a is b	a is the same object as b
a is not b	a is not the same object as b

Combinations are possible

```
In [70]: print( 1 < 2 <= 2.2 >= 2 > 1 )

True
```

'==' compares values while 'is' compares identities

```
In [74]: a = 1001     #an int-object with value 1001
         b = a        #the same object
         c = 1001     #a second object with value 1001

In [75]: print( a == b )
         print( a == c )

True
True
```

```
In [78]: print( a is b )
         print( a is c )

True
False
```

Warning: do not check equality of two floats (finite precision!!)

```
In [42]: from math import sin,pi
         print(sin(0)==0)
         print(sin(2*pi)==0)

True
False
```

```
In [81]: #instead of equality, test whether their difference is smaller than a tolerance value
         print( abs(sin(2*pi)-0) < 1e-8 )

True
```

In addition to int and float, many other data types can be compared as well:

```
In [79]: print(’color’==’color’)
         print(’color’==’colour’)
         print(’color 1’<’color 2’)

True
False
True
```

1.3.2 Boolean logic:

Operator	True, if
a and b	a and b are True
a or b	a or b (or both) are True
not a	a is False

```
In [85]: name = ’Bob’
         age = 98
         print( name==’bob’ and age<99)
         print( age < 99 and (age > 1 or name==’bob’) )

False
True
```

1.3.3 If . . . else . . .

```
In [101]: number_of_people = 3

          if number_of_people < 5:
              print(’Sorry, you need five or more people to play this game.’)

Sorry, you need five or more people to play this game.
```

```
In [100]: number_of_people = 6

          if number_of_people < 5:
              print(’Not enough people to play this game.’)
          else:
              print(’Thats enough. Enjoy!’)
```

```
Thats enough. Enjoy!
```

```
In [102]: number_of_people = 6

          if number_of_people < 5:
              print(’Not enough people to play this game.’)
          elif number_of_people < 10:
              print(’More would be better, but its sufficient.’)
          elif number_of_people < 20:
              print(’Perfect! Enjoy!’)
          elif number_of_people < 30:
              print(’Less would be better, but it will work somehow.’)
          else:
              print(’Sorry, but more than 30 is too much.’)
```

```
More would be better, but its sufficient.
```

Conditional expressions:

```
In [3]: x = 12

        #the long version:
        if x%2==0:
            message = "Even."
        else:
            message = "Odd."
        print(message)

        #the short version:
        print( "Even." if x%2==0 else "Odd." )
```

```
Even.
Even.
```

1.3.4 While ...

```
In [108]: value = 17

          while value < 21:
              print(value)
              value = value + 1
```

```
17
18
19
20
```

```
In [5]: value = 17
        max_value = 30
```

```
while True:
    value = value + 1
    if value > max_value:
        break              #stop here and escape the while loop
    elif value%2==0:
        continue           #stop here and continue the while loop
    print(value)
```

```
19
21
23
25
27
29
```

Warning: Make sure that the condition gets True after a finite number of steps!

```
In [118]: #Example of realy bad code:
          #The following code finishes if increment = 1 or 2.
          #But if increment = 0 or 3, the program is trapped in an infinite loop.
          #To stop it click on 'interrupt kernel'

          value = 0
          increment = 1

          while not value == 100:
              value = value + increment
```

1.3.5 Exercise: Find the smallest Fibonacci number which is bigger than 1000000.

Definition of Fibonacci numbers: $F_0 = 0$, $F_1 = 1$ and $F_n = F_{n-1} + F_{n-2}$
The first few numbers are: $\{0\ , 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots\}$

```
In [ ]:
```

1.4 Sequences & for-loops

1.4.1 Sequences

Sequence	mutable?	data type
list	yes	arbitrary
tuple	no	arbitrary
string	no	Unicode symbols

```
In [35]: a = [1,2,3,4,5]    #a list
         b = (1,2,3,4,5)    #a tuple
         c = '12345'        #a string
```

Since lists and tuples can contain arbitrary data types, they can be 'nested':

```
In [37]: nested_list = [[1,2,3],[4,5,6],[7,8,9]]
```

All three sequence types (tuples, strings and lists) share much of their syntax and functionality.

```
In [26]: print(len(a),len(b),len(c))
```

```
5 5 5
```

```
In [27]: print( a + a )
         print( b + b )
         print( c + c )
```

```
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
(1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
1234512345
```

single items are accessible by their index (starting from 0):

```
In [28]: print( a[0], b[1], c[2] )
```

```
1 2 3
```

Negative indices are counted from the end (starting with -1)

```
In [29]: print ( a[-1], b[-3] )
```

```
5 3
```

A subset of items can be accessed by “slices”.

Syntax: [I:J:K] means start from index I, stop at index J and take every K’th item. If I is omitted, start from the first item, if J is omitted, stop at the last item, and if K is omitted, take every item.

```
In [30]: print( a[1:4] ) #get items from 1 to 4
         print( a[3:5] ) #get items from 3 to 5
         print( a[:4] ) #get items from 0 to 4
         print( a[3:] ) #get items from 3 to the end
         print( a[::2] ) #get every second item
```

```
[2, 3, 4]
[4, 5]
[1, 2, 3, 4]
[4, 5]
[1, 3, 5]
```

The in-operator checks whether an item is in the sequence:

```
In [4]: 3 in [1,2,3,4,5]
```

```
Out[4]: True
```

```
In [5]: (2,3) in (1,2,3,4,5)
```

```
Out[5]: False
```

```
In [6]: 'cde' in 'abcdefgh'
```

```
Out[6]: True
```

In contrast to tuples and strings, lists are mutable. Items can be replaced, removed or added.

```
In [48]: a = [1,2,3,4]       #create list
         a[2] = 12           #replace item 2 by value 12
         a.append(34)        #add value 34 to the end
         a.extend([0,0,0])   #add several values to the end
         a.pop()             #remove last item
         a.insert(3, 'blub')#insert object before index 3
         a.reverse()         #reverse list
         print(a)
```

```
[0, 0, 34, 4, 'blub', 12, 2, 1]
```

1.4.2 For-loops

```
In [51]: numbers = [20,21,22,23]
         for i in numbers:
             print(i)
```

```
20
21
22
23
```

The iterations can be controlled with break and continue

```
In [56]: for i in numbers:
             if i%2==1:
                 continue
             print(i)
```

```
20
22
```

```
In [55]: for i in numbers[::2]:
             print(i)
```

```
20
22
```

For-loops can not only iterate through sequences, but also through ‘iterable’ objects, like range().

```
In [9]: #Example: We want to sum up all numbers betwen 0 and 100.
        #Instead of manually typing a list of all numbers, we can use range:
        s = 0
        for i in range(101):
            s = s + i
        print(s)
```

```
5050
```

List comprehensions: A short way to create a sequence.

```
In [13]: #long version: "for-loop"
         li = []
         for i in range(100):
             li.append(i*2)

         #short version:
         li = [2*i for i in range(101)]

         print(li)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52,
```

List comprehensions can be used as a filter:

```
In [12]: li = [2*i for i in range(101) if i%2==0]
         print(li)
```

```
[0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60, 64, 68, 72, 76, 80, 84, 88, 92, 96, 100,
```

Exercise: According to the Leibniz formula, π can be approximated by $\pi_N = 4\sum_{n=0}^{N} \frac{(-1)^n}{2n+1}$. Write a for loop which creates a list containing series π_N up to an arbitrary integer N. (hint: x^n is written in python as x**n or pow(x,n).)

```
In [ ]:
```

```
In [ ]:
```