Python 101
Data Structures

Overview

BRUNO VIEIRA

- Introduction to operators
- What are variables and data structures?
- What are the types available?
- What are the methods availabe for each type?
- What purpose each has?
- Application examples?

Arithmetic operators

- `+` Addition
- `-` Subtraction
- `*` Multiplication
- `**` Exponent
- `/` Division
- `//` Floor Division
- `%` Modulus

Arithmetic examples: + - * /

```
1  print 2 + 2
2  print 4 - 2
3  print 2 * 4
4  print 8 / 2
5  print 2 * 3 + 3 * 2
6  print (2 * 3) + (3 * 2)
7  print 2 * (3 + 3) * 2
8
```

```
4
2
8
4
12
12
24
```

Arithmetic examples: ** and operators style

```
1  print 2 ** 4
2  print 2**4
3  print 2 **4
4  print 2** 4
5
```

```
16
16
16
16
```

PEP 8 -- Style Guide for Python Code (http://www.python.org/dev/peps/pep-0008)

Arithmetic examples: / // in Python 2 vs Python 3

Python 2

```
1  print 8 / 3
2  print 8.0 / 3
3  print 8.0 // 3
5
```

```
2
2.6666666666666665
2
```

Python 3

```
1  print(8 / 3)
2  print(8.0 / 3)
3  print(8.0 // 3)
```

```
4
```

```
2.6666666666666665
2.6666666666666665
2
```

Python 2 vs Python 3

More information

Moving from Python 2 to Python 3 (cheat sheet) (http://ptgmedia.pearsoncmg.com/imprint_downloads/informit/promotions/python/python2python3.pdf)
Python 2 or Python 3 (http://wiki.python.org/moin/Python2orPython3)
What's New in Python (http://docs.python.org/py3k/whatsnew/index.html)

Arithmetic examples

```
1  print 8.0 / 3
2  print 8. / 3
3
```

```
2.6666666666666665
2.6666666666666665
```

Arithmetic examples

```
1  print 5.0 / 3
2  print 5.0 // 3
3  print 5.0 % 3
4
```

```
1.6666666666666667
1
2
```

Variables

"...a variable is a storage location and an associated symbolic name..." in Wikipedia (http://en.wikipedia.org/wiki/Variable_(computer_science))

Assignment statement

```
1  a = 1
2  print a
3
```

```
1
```

This is different from the equal operator `==`

```
1  a = 1
2  print a == 1
3  print a == 2
4
```

```
True
False
```

Variables usefulness

```
1  print "Interesting genes number: "
2  print 15
3  print "Uninteresting genes number"
4  print 30
5  print "Total genes: "
6  print 15 + 30
7
```

```
Interesting genes number:
15
Uninteresting genes number
30
Total genes:
```

```
45
```

Variables usefulness

```
1  igenes = 15 # Interesting genes
2  ugenes = 30 # Uninteresting genes
3  print "Interesting genes number: "
4  print igenes
5  print "Uninteresting genes number: "
6  print ugenes
7  print "Total genes: "
8  print igenes + ugenes
9
```

```
Interesting genes number:
15
Uninteresting genes number:
30
Total genes:
45
```

Variables forbidden names

The following identifiers are used as reserved words, or keywords of the language, and cannot be used as ordinary identifiers.

```
and       del       from      not       while
as        elif      global    or        with
assert    else      if        pass      yield
break     except    import    print     None
class     exec      in        raise     _*
continue  finally   is        return    __*__
def       for       lambda    try       __*
```

More information

Python Documentation: Identifiers (http://docs.python.org/reference/lexical_analysis.html#identifiers)

Assignment operators

- `=` Simple assignment operator

- `+=` Add AND assignment operator
- `-=` Subtract AND assignment operator
- `*=` Multiply AND assignment operator
- `/=` Divide AND assignment operator
- `//=` Floor Dividion and assigns a value
- `%=` Modulus AND assignment operator
- `**=` Exponent AND assignment operator

Assignment examples

```
1  a = 1
2  a = a + 3
3  print a
4
```

```
4
```

```
1  a = 1
2  a += 3
3  print a
4
```

```
4
```

Assignment examples

```
1  counter = 0
2  # Do something and increment counter
3  counter += 1
4  # Repeat something in a loop and increment counter
5  each time
6  counter += 1
7  # Check expected counter number reached and stop
8  working
   print counter
```

```
2
```

Assignment examples

```
a = 2
a *= 20; print a
a /= 3; print a
a //= 2; print a
a %= 4; print a
a **= 6; print a
```

```
40
13.333333333333334
6
2
64
```

Variables basic types

- Numbers (integers and floats)
- Strings

Integers and floats

```
a = 1
b = 2.0
print a + b
```

```
3
```

Strings assignment

```
a = "hello"
b = "world"
print a + b
```

```

```

```
helloworld
```

Strings assignment

Please try the following lines in the editor below

```
print "hello world"
print 'hello world'
print 'hello" "world'
print "hello' 'world"
print 'hello"
print "world'
```

```
1  print "hello world"
2
```

```
hello world
```

Strings assignment multiline

```
""" or '''
```

```
1  seq = """ABCD
2  EFGH
3  IJKLMNOPQ"""
4  print seq
5
```

```
ABCD
EFGH
IJKLMNOPQ
```

Integers + Strings

```
1  a = 1
2  b = "hello"
3  print a + b
4
```

```
TypeError: unsupported operand type(s) for Add: 'undefined'
and 'str'
```

Integers * Strings

```
1  a = 5
2  b = "hello "
3  print a * b
4
```

```
hello hello hello hello hello
```

Strings slices

```
variable = "string"
variable[start:stop:step]
```

```
1  seq = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
2  print seq[1:5]
3
```

```
BCDE
```

Strings slices

```
1  seq = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
2  print seq[:]
3  print seq[0]
4  print seq[0:5]
5  print seq[:5]
6  print seq[5:]
7
```

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
A
ABCDE
ABCDE
FGHIJKLMNOPQRSTUVWXYZ
```

Strings slices with negative indices

```
1  seq = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
2  print seq[-1]
3  print seq[-5:-1]
4  print seq[-0]
5
```

```
Z
VWXY
A
```

Strings slices with steps

```
1  seq = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
2  print seq[0:10:2]
3  print seq[-1:-11:-2]
4  print seq[:-11:-1]
5  print seq[::-1] # This one is very useful
6  print seq[::-2]
7
```

```
ACEGI
ZXVTR
ZYXWVUTSRQ
ZYXWVUTSRQPONMLKJIHGFEDCBA
ZXVTRPNLJHFDB
```

Strings slices out of range

```
1  seq = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
2  print seq[5:100]
3
```

```
FGHIJKLMNOPQRSTUVWXYZ
```

Strings methods

Strings methods

capitalize center count decode encode endswith expandtabs find format index isalnum isalpha isdigit islower isspace istitle isupper join ljust lower lstrip partition replace rfind rindex rjust rpartition rsplit rstrip split splitlines startswith strip swapcase title translate upper zfill

Total: 75

Strings methods

count endswith find islower isupper join lower lstrip replace rfind rsplit rstrip split splitlines startswith strip swapcase translate upper

Total: 19

Strings methods

- count
- find
- replace
- startswith and endswith
- islower and isupper
- lower, upper and swapcase
- join
- strip
- translate
- split and splitlines ← later with *Lists*

Strings methods

More information

Python Documentation (http://docs.python.org/library

/stdtypes.html#string-methods)

String count

```
str.count(sub[, start[, end]])
```

```
seq =
"TCCTGGAGGAGAATGGAGGTCAAGGGTCCAGCTGGAGAAGTTTAGGGTGTGGTG
GGGGTGA"
print seq.count("A")
```

```
13
```

```
seq =
"TCCTGGAGGAGAATGGAGGTCAAGGGTCCAGCTGGAGAAGTTTAGGGTGTGGTG
GGGGTGA"
print seq.count("A", 0 , 10)
```

```
2
```

String replace

```
str.replace(old, new[, count])
```

```
seq =
"TCCTGGAGGAGAATGGAGGTCAAGGGTCCAGCTGGAGAAGTTTAGGGTGTGGTG
GGGGTGA"
print seq.replace("T", "U")
```

```
UCCUGGAGGAGAAUGGAGGUCAAGGGUCCAGCUGGAGAAGUUUAGGGUGUGGUGGGGGUG
A
```

```
seq =
"TCCTGGAGGAGAATGGAGGTCAAGGGTCCAGCTGGAGAAGTTTAGGGTGTGGTG
GGGGTGA"
print seq.replace("T", "U", 3)
```

```
TCCTGGUGGUGUATGGAGGTCAAGGGTCCAGCTGGAGAAGTTTAGGGTGTGGTGGGGGTG
A
```

String translate

```
str.translate(table[, deletechars])
```

```
string.maketrans(from, to)
```

```
seq =
"TCCTGGAGGAGAATGGAGGTCGAGANNNNNGGTGTGGTGGGGGTGANNNNN"
print seq.translate(None, "AT")
```

```
CCGGGGGGGGGCGGGCCGCGGGGGGGGGGGGGGG
```

```
import string # We'll learn about this line later
seq =
"TCCTGGAGGAGAATGGAGGTCGAGANNNNNGGTGTGGTGGGGGTGANNNNN"
table = string.maketrans("AGTC", "1234")
print seq.translate(table, "N")
```

```
34432212212113221223421212232322322222321
```

String startswith and endswith

```
str.startswith(suffix[, start[, end]])
```

```
seq =
"TCCTGGAGGAGAATGGAGGTCAAGGGTCCAGCTGGAGAAGTTTAGGGTGTGGTG
GGGGTGA"
print seq.startswith("TCCT")
print seq.startswith("TGCT")
print seq.startswith("ATG", 12, 20)
```

```
True
False
True
```

```
seq =
"TCCTGGAGGAGAATGGAGGTCAAGGGTCCAGCTGGAGAAGTTTAGGGTGTGGTG
GGGGTGA"
print seq.endswith("TGA")
```

```
True
```

String find and rfind

```
str.find(sub[, start[, end]])
```

```
seq =
"TCCTGGAGGAGAATGGAGGTCAAGGGTCCAGCTGGAGAAGTTTAGGGTGTGGTG
GGGGTGA"
print seq.find("GTC")
print seq.rfind("GTC")
```

```
18
25
```

```
seq =
"TCCTGGAGGAGGTCAAGGGTCCAGCTGGAGAAGTTTAGGGTGTGGTGGGGGTGA
"
print seq.find("ATG", 0, 10)
```

```
-1
```

String islower and isupper

```
str.islower()
```

```
a = "T"
cod1 = "ATG"
cod2 = "AtG"
print a.isupper()
print cod1.isupper()
print cod2.isupper()
```

```
True
True
False
```

String lower, upper and swapcase

```
str.lower()
```

```
cod = "AtG"
print cod.lower()
```

```
print cod.upper()


```

```
atg
ATG
```

```
cod = "AtG"
print cod.swapcase()


```

```
aTg
```

String join

```
str.join(iterable)
```

```
str = "TCCTGG"
print ":".join(str)


```

```
T:C:C:T:G:G
```

String strip, lstrip and rstrip

```
str.strip([chars])
```

```
seq = "NNNNGCGCGCTGGAGGAGGTGAGAAGTTTAGGGTAAAAAAAAANNNN"
seq = seq.strip("N"); print seq
print seq.lstrip("GC")
print seq.rstrip("A")
print seq.strip("ATG")


```

```
GCGCGCTGGAGGAGGTGAGAAGTTTAGGGTAAAAAAAAA
TGGAGGAGGTGAGAAGTTTAGGGTAAAAAAAAA
GCGCGCTGGAGGAGGTGAGAAGTTTAGGGT
CGCGC
```

Data Structures

- Lists
- Tuples
- Sets
- Dicts

Lists assignment

```
[a, b, c]
```

```
1  a = [1,2,3,4,5]
2  b = ["a", "b", "c", "d", "f"]
3  print a[0]
4  print b[-1]
5
```

```
1
f
```

Lists assignment

```
1  a = [0] * 5
2  b = [1] * 5
3  c = ['NA'] * 5
4  d = range(5) # range([start], stop[, step]) [1]
5  print a
6  print b
7  print c
8  print d
9
```

```
[0, 0, 0, 0, 0]
[1, 1, 1, 1, 1]
['NA', 'NA', 'NA', 'NA', 'NA']
[0, 1, 2, 3, 4]
```

[1] Python Built-in Functions (http://docs.python.org/library/functions.html#built-in-functions)

Lists assignment

```
str.split([sep[, maxsplit]])
```

```
1  raw_data =
2  "TCCTGGAGGAG;GTCAAGGGTCCAGCT;GGAGAAGTTTAGGG;TGTGGTG;GGG
3  GTGA"
4  sequences = raw_data.split(";")
   print sequences
```

```
['TCCTGGAGGAG', 'GTCAAGGGTCCAGCT', 'GGAGAAGTTTAGGG',
'TGTGGTG', 'GGGGTGA']
```

```
1  raw_data =
2  "TCCTGGAGGAG;GTCAAGGGTCCAGCT;GGAGAAGTTTAGGG;TGTGGTG;GGG
4  GTGA"
   sequences = raw_data.split(";", 3)
```

```
['TCCTGGAGGAG', 'GTCAAGGGTCCAGCT',
'GGAGAAGTTTAGGG;TGTGGTG;GGGGTGA']
```

Lists assignment

```
str.splitlines([keepends])
```

```
1  raw_data = """TCCTGGAGGAG
2  GTCAAGGGTCCAGCT
3  GGAGAAGTTTAGGG"""
4  sequences = raw_data.splitlines()
6
```

```
['TCCTGGAGGAG', 'GTCAAGGGTCCAGCT', 'GGAGAAGTTTAGGG']
```

Lists assignment

```
str.splitlines([keepends])
```

```
1  raw_data = """TCCTGGAGGAG
2  GTCAAGGGTCCAGCT
```

```
GGAGAAGTTTAGGG"""
sequences = raw_data.splitlines(True)
```

```
['TCCTGGAGGAGn', 'GTCAAGGGTCCAGCTn', 'GGAGAAGTTTAGGG']
```

Multi-dimensional lists

```
mylist = [["A", "T", "G", "C"], [1, 2, 3, 4]]
print mylist
print mylist[0]
print mylist[1][2]
```

```
[['A', 'T', 'G', 'C'], [1, 2, 3, 4]]
['A', 'T', 'G', 'C']
3
```

Lists methods

- count
- index
- append
- insert
- remove
- pop
- reverse
- sort

More information

Python Documentation (http://docs.python.org/library/stdtypes.html#mutable-sequence-types)

Lists count, index, append and insert

```
mylist = ["A", "T", "G", "C", "A", "T"]
print mylist.count("A")
print mylist.index("C")
```

```
4  mylist.append("TAIL")
5  mylist.insert(3, "MIDDLE")
6  print mylist
7
```

```
2
3
['A', 'T', 'G', 'MIDDLE', 'C', 'A', 'T', 'TAIL']
```

Lists remove

```
1  mylist = ["A", "T", "G", "C", "A", "T"]
2  mylist.remove("A")
3  print mylist
4  mylist.remove("A")
5  print mylist
6
```

```
['T', 'G', 'C', 'A', 'T']
['T', 'G', 'C', 'T']
```

Lists pop

```
1  mylist = ["A", "T", "G", "C", "A", "T"]
2  mylist.pop(2)
3  print mylist
4  print mylist.pop(1)
5  print mylist
6
```

```
['A', 'T', 'C', 'A', 'T']
T
['A', 'C', 'A', 'T']
```

Lists reverse and sort

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

```
mylist.sort()
print mylist
mylist.reverse()
print mylist

```

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

Lists with Python Built-in Functions

```
mylist=range(5)
print mylist
print len(mylist)
print min(mylist)
print max(mylist)

```

```
[0, 1, 2, 3, 4]
5
0
4
```

More information

Python Documentation (http://docs.python.org/library/functions.html#built-in-functions)

Lists with join

```
mylist = ["A", "T", "G", "C", "A", "T"]
print "".join(mylist)

```

```
ATGCAT
```

Tuples

- Lists that are immutable (likes strings)

- Useful for storing heterogeneous data in which order has semantic value (like coordinates)
- Fast!

More information

Python Docs: Tuples and Sequences (http://docs.python.org/tutorial/datastructures.html#tuples-and-sequences)

Tuples assignment

```
1  coord1 = 12, 35 # pair of coordinates
2  coord2 = (32, 12)
3  coordinates = [coord1, coord2]
4  print coordinates
5
```

```
[(12, 35), (32, 12)]
```

Sets

- Unordered collection with no duplicate elements
- Uses include membership testing and eliminating duplicate entries
- Also support mathematical operations like union, intersection, difference, and symmetric difference.

More information

Python Docs: Sets (http://docs.python.org/tutorial/datastructures.html#sets)

Sets

```
1  palette = set(['blue', 'red', 'green', 'red'])
2  print "blue" in palette
3  print "magenta" in palette
4
```

```
True
False
```

Sets

```
1  p1 = set(['blue', 'red', 'green', 'red'])
2  p2 = set(['yellow', 'green', 'blue', 'yellow', 'blue'])
3  print p1 - p2 # colors in p1 but not in p2
4  print p1 | p2 # colors in either p1 or p2
5  print p1 & p2 # colors in both p1 and p2
6  print p1 ^ p2 # colors in p1 or p2 but not both
8
```

```
set(['red'])
set(['blue', 'green', 'yellow', 'red'])
set(['blue', 'green'])
set(['red', 'yellow'])
```

Dictionaries

- Unordered set of "key: value" pairs, with the requirement that the keys are unique
- Known in other languages as "associative memories" or "associative arrays"
- Indexed by keys (unlike sequences, which are indexed by a range of numbers)
- Indices can be any immutable type (strings, numbers, or tuples of immutable objects)
- Usefull for storing, extracting or deleting values with a key

Dictionaries assignment

```
{'a': 1, 'b': 2, 'c': 3}
{'key1': "Value", 'key2': "Value", 1: "Another Value", 'd': 42}
```

```
1  sequences = {'s1': "AGTAGCGT", 's2': "ATGAC",
2  'primer': "AGCTGCTAG"}
3  print sequences['primer']
4  del sequences['s2']
5  print sequences
```

```
AGCTGCTAG
{'s1': 'AGTAGCGT', 'primer': 'AGCTGCTAG'}
```

Dictionaries assignment

```
1  sequences = {'s1': "AGTAGCGT", 's2': "ATGAC",
2  'primer': "AGCTGCTAG"}
3  sequences['s1'] = "AAAAAAAA"
4  print sequences
5  print sequences.keys()
6  print 'primer' in sequences
```

```
{'s1': 'AAAAAAAA', 's2': 'ATGAC', 'primer': 'AGCTGCTAG'}
['s1', 's2', 'primer']
True
```

Wrap up

- Arithmetic operators `+ - * / // % **`
- Assignment operators `+= -= *= /= //= %= **=`
- Numbers and Strings `a = 1; b = "Hello World"`
- String methods `count, find, join, translate, split, etc`
- Lists and methods `a = [1, 2]; append, pop, reverse, sort, etc`
- Some Built-in functions `range, len, min, max`
- Tuples `a = 1, 2, 3; b = (1, 2, 3)`
- Sets `a = set([1, 2, 3])`
- Dictionaries `a = {a: 1, b: 2, c: 3}`