

The Python Conceptual Hierarchy

- 1. Programs are composed of modules.
- 2. Modules contain statements.
- 3. Statements contain expressions.
- 4. Expressions create and process **objects**.



Built-in objects

- ... make programs easy to write
- ... are components of extensions
- ... are often more efficient than custom data structures
- ... are a standard part of the language

Core Data Types

Types and examples

- Numbers
- Strings
- Lists
 - Dictionaries
- Tuples
- Files
- Sets
- Other core types
- Program unit types
 - Implementation-related types

```
123, 3.14, 3+4j, Decimal()

'spam', "Hello World!"

[1, [2, 'three'], 4.5]

{'food': 'spam', 'taste': 'yum'}

(1, 'spam', 4, 'U')

open('eggs.txt')

set('abc'), {'a', 'b', 'c'}

Booleans, types, None

Functions, modules, classes

Compiled code, stack tracebacks
```



Core Data Types: Numbers

Basics

```
>>> 123 + 222
345
>>> 1.5 * 4
6.0
>>> 2 ** 100
1267650600228229401496703205376
>>> import math
>>> math.pi
3.141592653589793
>>> math.sqrt(85)
9.219544457292887
```

Core Data Types: Strings

Sequence oparations

```
>>> S = 'Spam'
>>> len(S)
4
>>> S[0]  # index from the first item
'S'
>>> S[1]
'p'
>>> S[-1]  # index backward
'm'
>>> S[-2]
'a'
>>> S[1:3]  # range
'pa'
```



```
>>> S[1:]
'pam'
>>> S
'Spam'
>>> S[0:3]
'Spa'
>>> S[:3]
>>> S + 'xyz' # Concatenation
'Spamxyz'
>>> S # S is unchanged
'Spam'
>>> S * 8 # Repetition
'SpamSpamSpamSpamSpamSpamSpam'
```



```
>>> S
'Spam'
# Immutable objects cannot be changed
>>> S[0] = 'z'
( . . . )
TypeError: 'str' object does not support item
assignment
# But we can run expressions to make/reassign new
objects to an object-name:
>>> S = 'z' + S[1:]
>>> S
'zpam'
```



String manipulator methods

```
>>> S.find('pa')
1

>>> S.replace('pa', 'XYZ')
'SXYZm'
>>> S
'Spam'

>>> line = 'aaa,bbb,cccc,dd'
>>> line.split(',')
['aaa', 'bbb', 'cccc', 'dd']
```



String manipulator methods

```
>>> S = 'spam8'
>>> S.upper()
'SPAM8'
>>> S.isalpha()
                                   >>> S.isalnum()
False
                                              True
>>> line = 'aaa,bbb,cccc,dd\n'
# Remove whitespace characters on the right side
>>> line.rstrip()
'aaa,bbb,cccc,dd'
# Combine two operations
>>> line.rstrip().split(',')
['aaa', 'bbb', 'ccccc', 'dd']
```



String manipulator methods

```
>>> '%s, eggs, and %s' % ('spam', 'SPAM!')
'spam, eggs, and SPAM!' # Formatting expression (all)
>>> '{0}, eggs, and {1}'.format('spam', 'SPAM!')
'spam, eggs, and SPAM!' # Formatting method (2.6+,
3.0+)
>>> '{}, eggs, and {}'.format('spam', 'SPAM!')
'spam, eggs, and SPAM!'

>>> '{:,.2f}'.format(296999.2567)
'296,999.26'
>>> '%.2f | %+05d' % (3.14159, -42)
'3.14 | -0042'
```



Regular expressions

```
>>> import re
>>> match = re.match('Hello[ \t]*(.*)world', 'Hello
Python world')
>>> match.group(1)
'Python '
>>> match = re.match('[/:](.*)[/:](.*)[/:](.*)',
'/usr/home:lumberjack')
>>> match.groups()
('usr', 'home', 'lumberjack')
>>> re.split('[/:]', '/usr/home/lumberjack')
['', 'usr', 'home', 'lumberjack']
```



Sequence Operations

```
>>> L = [123, 'spam', 1.23]
>>> len(L)
                   # Indexing by position
>>> L[0]
123
>>> L[:-1]
                  # Slicing a list returns a new list
[123, 'spam']
>>> L + [4, 5, 6] # Concat/repeat make new lists too
[123, 'spam', 1.23, 4, 5, 6]
>>> T<sub>1</sub> * 2
[123, 'spam', 1.23, 123, 'spam', 1.23]
>>> L
[123, 'spam', 1.23]
```

Core Data Types: Lists

Type-Specific Operations

```
>>> L.append('NI')
>>> L
[123, 'spam', 1.23, 'NI']
>>> L.pop(2)
1.23
>>> L
[123, 'spam', 'NI']
>>> M = ['bb', 'aa', 'cc']
>>> M.sort()
>>> M
['aa', 'bb', 'cc']
>>> M.reverse()
>>> M
['cc', 'bb', 'aa']
```

Core Data Types: Lists

Nesting



A powerful way to process structures.

```
>>> M

[[1, 2, 3], [4, 5, 6], [7, 8, 9]]

>>> col2 = [row[1] for row in M]

>>> col2

[2, 5, 8]
```

Add 1 to each item in column 2 >>> [row[1] + 1 for row in M] [3, 6, 9]



Core Data Types: Lists

Comprehensions

```
# Filter out odd items
>>> [row[1] for row in M if row[1] % 2 == 0]
[2, 8]
```

```
# Collect a diagonal from matrix
>>> diag = [M[i][i] for i in [0, 1, 2]]
>>> diag
[1, 5, 9]
```

```
# Repeat characters in a string
>>> doubles = [c * 2 for c in 'spam']
>>> doubles
['ss', 'pp', 'aa', 'mm']
```



Core Data Types: Lists

Comprehensions

```
# 0..3 (list() required in 3.X)
>>> list(range(4))
[0, 1, 2, 3]

# -6 to +6 by 2 (need list() in 3.X)
>>> list(range(-6, 7, 2))
[-6, -4, -2, 0, 2, 4, 6]

# Multiple values
>>> [[x ** 2, x ** 3] for x in range(4)]
[[0, 0], [1, 1], [4, 8], [9, 27]]
```



Core Data Types: Dictionaries

- Not sequences, instead mappings
- Key value pairs
- Relative position doesn't matter
- Mutable (may be changed in place)
- Can grow and shrink on demand

Core Data Types: Dictionaries

Mapping Operations

```
>>> D = {'food': 'Spam', 'quantity': 4}
     >>> D['food']
'Spam'
     >>> D['quantity'] += 1
>>> D
{'food': 'Spam', 'quantity': 5}
     >>> D = \{ \}
>>> D['name'] = 'Bob'
>>> D['job'] = 'dev'
```

Core Data Types: Dictionaries

Nesting

```
>>> rec = {'name': {'first': 'Bob', 'last':
'Smith'},
                'jobs': ['dev', 'mgr'],
                'age': 40.5}
    >>> rec['name']
     {'last': 'Smith', 'first': 'Bob'}
    >>> rec['name']['last']
     'Smith'
```



Core Data Types: Tuples

A list, that can not be changed.

- Pronounced "toople" or "tuhple,"
- Sequences, like lists, but they are immutable, like strings
- Used to represent fixed collections of items
- Have type-specific callable methods, but not as many as lists have
- Support mixed types and nesting, but they don't grow and shrink (because they are immutable)



A list, that can not be changed.

```
>>> T = (1, 2, 3, 4)
>>> len(T)
4

>>> T[0]
1

>>> T.index(4)
3

# Tuples are immutable
>>> T[0] = 2
TypeError: 'tuple' object does not support item assignment
```

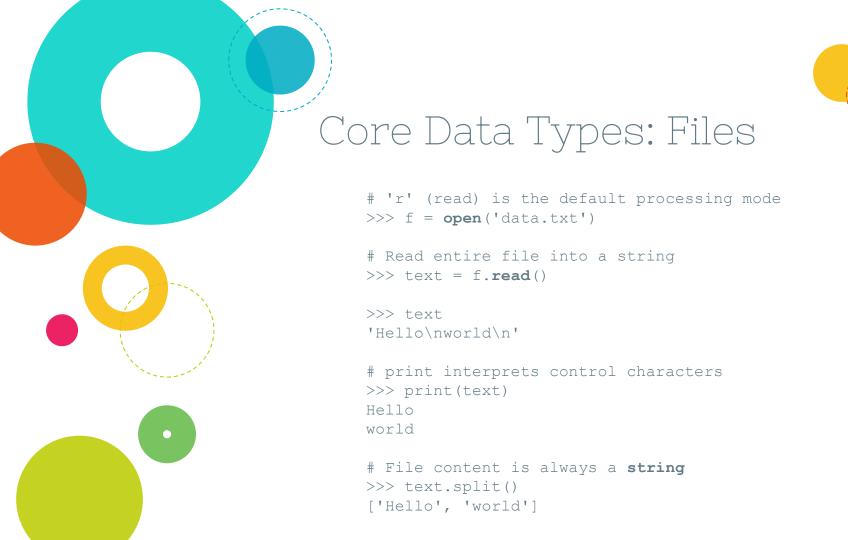


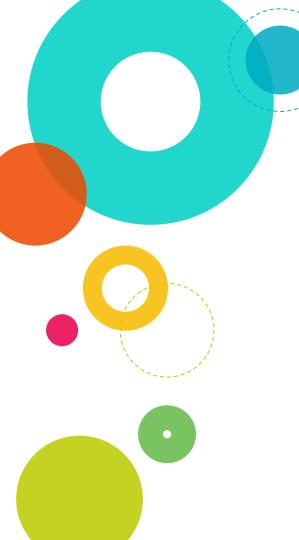
```
# Make a new file in output mode ('w' is write)
>>> f = open('data.txt', 'w')

# Write strings of characters to it
>>> f.write('Hello\n')
6

# Return number of items written in Python 3.X
>>> f.write('world\n')
6

# Close to flush output buffers to disk
>>> f.close()
```





Homework

- 1. Try out all commands learned (also try to change parametring, and see the effects)
- 2. Write program
 - After start ask user of input with the command: text = input("Question comes here")
 - Print out the text entered with UPPERCASES
 - Print out the number of characters entered
 Call following to get the length of the string: