

Python3 Builtins

Technologies for Big Data
with PYTHON

marco milanesio
MS DATA SCIENCE 2019-2020

Personal productivity tool

- Python is a tool
 - Munging
 - Cleaning
 - Counting
 - Organizing
- Secret weapons
 - tuple, list, set, dict
 - collections module
 - builtin operations

Everywhere

- The builtin types are always available:
 - All versions of Python
 - All operating systems
 - All distributions of Python
- You don't have to install anything

Flexibility

- Dynamic typing
 - You can make data structures out of anything (almost)
 - You barely need to think about it
 - It just "works"
- *If it walks like a duck and it quacks like a duck, then it must be a duck.*

Performance

- Builtin types are fast for coding
 - Memory is cheap
 - CPU cycles are cheap
 - Your time is expensive
- They provide a basic foundation for exploring ideas
- Optimize later (if needed)

Easiness

- Abstract away "annoying" details (memory)
- You still have to think
- Personal productivity

- The Zen of Python
 - import this

Builtin types

Tuple ()

- Record, Structure
- Packing and unpacking
- A row in a database

```
record = (val1, val2, val3)  
a, b, c = record  
val = record[0]
```

- `collections.namedtuple`

List []

- Mutable sequence, Array
- Enforcing order

```
items = [val1, val2, val3]
x = items[2]
items[0] = x
del items[1]
items.append(value)
items.sort()
new_items = sorted(items)
```

Set {}

- Set
- Uniqueness, membership tests

```
s = {val1, val2, val3}  
s.add(val4)  
s.remove(val2)  
val in s
```

Dict {::}

- Mapping, Associative array
- Lookup tables, indices

```
d = {key1: val1, key2:val2, key3:val3}
val = d[key]
d[key] = val
del d[key]
key in d
```

defaultdict

- `collections.defaultdict`
- Multidicts, one-to-many relationships
- Grouping

```
d = defaultdict(list)
d[key].append(value)
values = d[key]

d = defaultdict(set)
d[key].add(value)
unique_values = d[key]
```

- Preserve order of keys

Counter

- `collections.Counter`
- Counting, histograms, tabulations

```
c = Counter(sequence)
c[key] += n
c.most_common(n)
```

Powers

Basic powers

- Iterations

```
for item in sequence:  
    ...
```

- Variants

```
for pos, item in enumerate(sequence):  
    ...  
for x, y in zip(sequence1, sequence2):  
    ...
```

- Reductions

```
sum(sequence)  
min(sequence)  
max(sequence)  
any(sequence)  
all(sequence)
```

Super-powers

- List comprehension

```
[ expr for x in iterable if condition ]
```

- Set comprehension

```
{ expr for x in iterable if condition }
```

- Dict comprehension

```
{ k:v for k,v in iterable if condition }
```


Iter-powers

- Generator expression

```
( expr for x in iterable if condition )
```

- Combined with reduction

```
sum(expr for x in iterable if condition)
```

- PRO-TIP: generator expressions allow you to process HUGE amounts of data incrementally saving tons of memory!
- Awesome to feed loops...