Builtin `dict` & `**kwargs` preserve *some* order

Python[*] 3.6+

[*] The C based Python 3.6+ reference implementation and PyPy 4+ just do it, and so can {{*YourOtherImplementation*}} 🙄

Stacks and Queues

- Real world 'printed' dictionaries expose sorted keys
- Topic of talk is **stable ordering** and (**not** sorting)
  - Focus is on observable behavior of keys (*and sets*)
- Iff key order preserved (*by underlying hash mapping*),

  then thoughtful creation of a `dict` say `d` may allow:
  - Queue: `for k in d.keys():` # 😄
  - Stack: `for k in reversed(tuple(d)):` # 😄

Proverbs / Common sense facts we learn when growing up

OK, carved into brains (know the *fetters of your mind*):

☝ You can't **have** your cake and **eat it** 🗣!

— *Educational Person a.k.a. Life*™

Now is this what you wanted? Like: ☝ Two for one 🗣!

of the "*Local Brain Sales Rep.*" ... or another variant of:
🕵 **blocking our views through artificial rules**?

First Learn, Second Follow, Third "(Reverse) Learn" 🏃🏢👮

Common sense facts: Base of Culture or only Hear Say?
🤔 One such *fact* learned the hard way by *most of us* is:

The native Python dict does not preserve insert order.

Python 3.6+ builtin hash maps preserve insert order!
☝ … `dict`, `set` and `**kwargs` (PEP 468 implemented).

PEP 468 ⇒ "Preserving the order of `**kwargs` in a function" 😄 so, *we now can have our cake and eat it too?*

Question: PEP 468: "`**kwargs` order" - Rely on it or not?

- Yes! Use cases (from PEP 468):
  - print out key:value pairs in CLI output
  - map semantic names to column order in a CSV
  - serialise attributes and elements in particular orders in XML
  - serialise map keys in particular orders in human readable formats like JSON and YAML.

Question: New dict implementation - Rely on it or not?

- The dict type now uses "compact" representation [...]
  - Memory usage between 20% - 25% smaller ≪ v3.5
- The order-preserving aspect [...] considered an implementation detail and should not be relied ...
  - This may change in the future, but it is desired [...] a few releases before changing the language spec to mandate order-preserving semantics for all current and future Python implementations [...] .

(1/7) 🤔 Explore the good news and our *bright* future

Short interactive session - you're free to ignore 🙄:

```
Python 3.6.2 (default, Jul 17 2017, 16:44:47)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.42.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> d = {'foo': 1, 'bar': 2, 'baz': 3}
>>> for k, v in d.items():
...     print(k, "->", v)
```

```
# For now an implementation detail ;-)
foo -> 1
bar -> 2
baz -> 3
```

(2/7) Update a key's value

Iteration shows, value update preserves key position:

```
>>> d['foo'] = 42
>>> for k, v in d.items():
...     print(k, "->", v)

foo -> 42
bar -> 2
baz -> 3
```

(3/7) Delete the key (position now *taken* from next!)

```
>>> del d['foo']
>>> for k, v in d.items():
...     print(k, "->", v)

bar -> 2
baz -> 3
```

(4/7) "Re-Insert" (kind of) removed key with some value

```
>>> d['foo'] = -1
```

But **now** `'foo: -1'` is **appended** (insert order!), so:

```
>>> for k, v in d.items():
...     print(k, "->", v)

bar -> 2
baz -> 3
foo -> -1
```

(5/7) Short *dirty* check to show off PEP 468

```
>>> # Remember: **d ⟼ bar=2, baz=3, foo=-1
>>> print(**d)  # HACK A DID ACK
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'bar' is an invalid keyword argument for this function
            ^^^
```

⇒ ☝ Order preserved; Python 2.7.13 on OS X raises:

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'baz' is an invalid keyword argument for this function
            ^^^
```

(6/7) Some other function exposing PEP 468 behavior

```
>>> def a_stack(pos, *args, **kwargs):
...     """Now for something completely different ..."""
...     for k in reversed(tuple(kwargs)::
...             print(k, "->", kwargs[k])
...
>>> # Remember: **d ⟼ bar=2, baz=3, foo=-1
>>> a_stack(True, **d)

foo -> -1
baz -> 3
bar -> 2
```

(7/7) The builtin set now also preserves order

```
>>> # Remember: **d ⟼ bar=2, baz=3, foo=-1
>>> s = set(d.keys())  # Using set constructor
>>> print(tuple(s))

('bar', 'baz', 'foo')  # Also an implementation detail ;-)

>>> s = {'bar', 'baz', 'foo'}  # Fresh set literal
>>> print(tuple(s))

('bar', 'baz', 'foo')  # Dito implementation detail ;-)
```

What gives?

… still **not** clear what this ***means***,

but will notice - *as time goes by* …

Any questions? Thoughts?

-- Thanks!