Repetition

January 20, 2020

1 Repetition

1.1 Objects and variables

- All data in Python are objects
- Every object has a value, a data type and a unique ID
- Variables are names (identifiers) that refer to objects
- A variable can only refer to one object at a a time
- Assignment (=) makes a variable refer to a different object
- All data types have constructors, that attempt to coerce its argument to the data type
  - int(4.0) -> 4
  - int('4') -> 4
  - int('four') -> ValueError

1.2 Data types

1.2.1 Simple data types

- Numeric data types:
  - int - unlimited size
  - float - IEEE 457
  - complex - two floats - real, imaginary
- Sequences:
  - str - Unicode code points - 1-4 bytes
  - bytes - 8 bits - often used for data import/export

1.2.2 Collection data types

- Mutable
  - list - sequence of objects
  - dict - collection of key -> value
  - set - a non ordered collection of uinque elements
- Non mutable
  - tuple - a sequence of objects

1.3 Operators

1.3.1 Numeric types

- Numeric operators, i precedence order - (), **, ..., , / //, %, +, -, ..., =, +=, =

1.3.2 Sequence types

- Indexing - indicies start at 0 - a[2]
- Slicing - [start] : [first element not included] (: step)
  - a[1:3] - element 1, 2
  - a[5:10:2] - element 5, 7, 9
  - a[10:4:-2] - element 10, 8, 6
  - a[3:] - element 3 to the end of the sequence
  - a[:5] - fom the start to and including element 4
  - a[::-1] - reverse the sequence object
  - a[:] - copy the sequence object

2 Numeric types

[1]:
```
# Integers
i = 1
i += 2918340567787979789879 9
i
```

[1]:
```
2918340567787979789880 0
```

[2]:
```
# Floats
i += 1.0
i
```

[2]:
```
2.91834056778798e+22
```

[8]:
```
import math
math.sqrt(i)
```

[8]:
```
170831512543.4409
```

3 Strings

[4]:
```
sentence = 'Gurkorna cyklar i motvind'
sentence[3]
```

[4]:
```
'k'
```

[9]:
```
sentence[:6]
```

[9]:
```
'Gurkor'
```

[10]:
```
sentence[-4:]
```

[10]:
```
'vind'
```

[6]:
```
sentence[::-1]
```

[6]:
```
'dnivtom i ralkyc anrokruG'
```

[11]:
```
sentence [::-2]
```

[11]:
```
'ditmirly norG'
```

[12]:
```
sentence.upper()
```

[12]:
```
'GURKORNA CYKLAR I MOTVIND'
```

4 Lists

[13]:
```
lucky_numbers = [5, 19, 47, -293]
dog_names = ['fluffy', 'fido', 'pluto', 'buster', 'kitty']
senseless = [lucky_numbers, dog_names]
senseless
```

[13]:
```
[[5, 19, 47, -293], ['fluffy', 'fido', 'pluto', 'buster', 'kitty']]
```

[14]:
```
lucky_numbers[2]  # Single object
```

[14]:
```
47
```

[20]:
```
lucky_numbers[2:3]  # List with 1 element
```

[20]:
```
[47]
```

5 Dictionaries

[16]:
```
scores = {0 : 'Abysmal', 1: 'Lousy', 2: 'Almost ok', 3: 'Fine', 4: 'Good', 5:␣
↪'Fantastic'}
scores[2]
```

[16]:
```
'Almost ok'
```

[18]:
```
reverse_scores = {}
for key, val in scores.items():
    reverse_scores[val] = key

reverse_scores
```

[18]:
```
{'Abysmal': 0,
 'Lousy': 1,
```

```
 'Almost ok': 2,
 'Fine': 3,
 'Good': 4,
 'Fantastic': 5}
```

[19]:
```
reverse_scores['Fantastic']
```

[19]:
```
5
```

[28]:
```
reverse_scores2 = {val: key for key, val in scores.items()}
reverse_scores2
```

[28]:
```
{'Abysmal': 0,
 'Lousy': 1,
 'Almost ok': 2,
 'Fine': 3,
 'Good': 4,
 'Fantastic': 5}
```

6 Never use mutable defaults

[27]:
```
def myfun(mylist=[]):
    mylist.append(1)
    print(mylist)

myfun()
myfun()
```

```
[1]
[1, 1]
```

7 Do this instead

[31]:
```
def myfun2(mylist=None):
    if not mylist:
        mylist = [1]
    print(mylist)

myfun2()
myfun2()
```

```
[1]
[1]
```

8 Function as argument with lambda

[33]:
```
costs = [('b', '11'), ('a', '10.3'), ('x', '-2')]
sorted(costs, key=lambda x: float(x[1]))
```

[33]:
```
[('x', '-2'), ('a', '10.3'), ('b', '11')]
```

9 Function as argument without lambda

[34]:
```
def mysort(x):
    return (float(x[1]))

sorted(costs, key=mysort)
```

[34]:
```
[('x', '-2'), ('a', '10.3'), ('b', '11')]
```

10 What is an object?

A set of data, and operations on those data.

11 What is a class?

A template for creating objects of a type - user defined type

[38]:
```
class Vehicle:
    pass

class Tank(Vehicle):
    pass

x = Tank()
isinstance(x, Tank)
```

[38]:
```
True
```

[39]:
```
isinstance(x, Vehicle)
```

[39]:
```
True
```

12 Skipped subjects

12.1 Functions

- Asyncronous functions
- Generator functions

12.2 Object oriented programming

- Multiple inheritance
- Class methods
- Static methods

[56]:
```
import datetime
now = datetime.datetime.now()
print(type(now))
print(now.strftime('%Y-%B-%d'))
```

```
<class 'datetime.datetime'>
2019-November-13
```

[90]:
```
import time
from functools import wraps
def timeit(target_function):
    @wraps(target_function)
    def wrapper(*args, **kwargs):
        before = time.time()
        result = target_function(*args, **kwargs)
        after = time.time()
        print(after-before)
        return result
    return wrapper

@timeit
def myfun():
    time.sleep(3)

# myfun = timeit(myfun)

myfun()
```

```
3.0028114318847656
```

[91]:
```
myfun.__name__
```

[91]:
```
'myfun'
```

[ ]:
```
class MyException(ValueError):
    pass

try:
    pass
except MyException as e:
    #...
```

[81]:
```
with open('simple.txt') as fp:
    lines = fp.readlines()
    lines = [line.rstrip() for line in lines]
    lines = [line for line in lines if line and not line.startswith('#')]
lines
```

[81]:
```
['Line 1', 'Line 2', 'Line 3', 'Line 4']
```

[84]:
```
import re
pattern = r'^.*2019-1[01].*gnome.+'
with open('/var/log/alternatives.log') as logfile:
    lines = logfile.readlines()
    lines = [line for line in lines if re.match(pattern, line)]

for line in lines:
    print(line)
```

```
update-alternatives 2019-11-03 06:38:48: run with --install /usr/bin/gnome-www-
browser gnome-www-browser /usr/bin/firefox-esr 70 --slave
/usr/share/man/man1/gnome-www-browser.1.gz gnome-www-browser.1.gz
/usr/share/man/man1/firefox-esr.1.gz

update-alternatives 2019-11-05 19:24:10: run with --install /usr/bin/gnome-www-
browser gnome-www-browser /usr/bin/google-chrome-stable 200

update-alternatives 2019-11-05 19:24:11: run with --install /usr/bin/gnome-www-
browser gnome-www-browser /usr/bin/vivaldi-stable 200
```