

String

message = 'Hello world' } result is Hello
print(message[1:5])
print(message.count('l')) → how many time does l exist in var message
print(message.find('world')) → if can't find, will return -1 as result.
message.replace('oldword', 'newworld')
formatting: greeting, name = 'Hello', 'Ibrahim'
→ message = greeting + '!' + name + '. Welcome'
→ message = '{},{}.Welcome'.format(greeting, name)
→ message = f'{greeting},{name}.Welcome'
print(dir(message)) → show all method that can be used in message var
print(help(str)) → get .. in details
print(help(str.lower())) → show what lower() does.

Lists

Courses = ['php', 'python', 'security', 'RHCE']
Courses.append('js') → add
Courses.insert(0, 'CSS') → add to specific location in list
Courses.extend(new_list) → add new list to courses list value individual
Courses.remove('php')
Courses.pop('html') → remove, by default will remove last value.
Course.reverse()
Course.sort()
Courses.sort(reverse=True) → sort & reverse in one line.
sorted_courses = sorted(courses) → it's useful because sometime we want to alter our main
courses.index('php') → where php value exists
Courses_str = '-'.join(courses)
new_list = course_str.split('-') → results as a list

tuples and list are exact the same, but list add be edited tuples can't

Sets

Set will remove duplicate value

Courses = {'php', 'js', 'python'}
new_courses = {'php', 'js', 'Ruby', 'C++'}
print(Courses.intersection(new_courses)) → Show similar Courses
print(Courses.difference(new_courses)) → Show difference Courses
print(Courses.union(new_courses))

How to create

```
2 # Empty Lists
3 empty_list = []
4 empty_list = list()
5
6 # Empty Tuples
7 empty_tuple = ()
8 empty_tuple = tuple()
9
10 # Empty Sets
11 empty_set = {} # This isn't right! It's a dict
12 empty_set = set()
```

Dicks

Student = { 'name': 'Ibrahim', 'age': 20 }

print(Student['name']) → get value of name key, not exist? return errorkey

print(Student.get('name')) → get ~ ~ ~ ~ ~ return None

print(Student.get('name', 'Not Found')) → raise not found, if key not exist

Student['name'] = 'Ahmed' update one key.

Student.update({ 'name': 'Karim', 'age': 18 }) update multiple keys.

del Student['name'] → delete

age = Student.pop('age') → delete age, but save value in var age (pop will remove last also will return that val)

Student.keys()

Student.values()

Student.items() → print key + values

for key, value in Student.items():

print(key, value)

Additional Note:

```

2 # False Values:
3 # False
4 # None
5 # Zero of any numeric type
6 # Any empty sequence. For example, '', (), []
7 # Any empty mapping. For example, {}
8
9 condition = 10 ➤ anything else will be True
10
11 if condition:
12     print('Evaluated to True')
13 else:
14     print('Evaluated to False')
15

```

Range (start, end)

Fractions

```

def student_info(*args, **kwargs):
    print(args)
    print(kwargs)

student_info('Math', 'Art', name='John', age=22)

```

```

('Math', 'Art')
{'name': 'John', 'age': 22}
[Finished in 0.0s]

```

```

3 def student_info(*args, **kwargs):
4     print(args)
5     print(kwargs)
6
7 courses = ['Math', 'Art']
8 info = {'name': 'John', 'age': 22}
9
10 student_info(*courses, **info)
11
12 ➤ for list
13 ➤ for dict

```

Models

How to set a pythonpath environment variable

Sys.path → will show you all the python environment variable
in Mac's open terminal → nano ~/.bash-profile → and set following
 export PYTHONPATH="your module path" + restart terminal

in Windows: System info window → advanced system setting → environment variable

→ new → name = PYTHONPATH

Value = your module path

In any python file & sys.path.append('your module python path')

LEGB (Local, Enclosing, Global, Built-in)

Top level in a module

- $x = 'global x'$ → global variable
 - def test():
 - Y = 'local y' → local variable, only in def is defined
 - global x → global ~
 - def test(x): → x variable is local in the function
 - print(x)
 - import builtins
 - print(dir(builtins)) → will show all builtin variables.
- Compiler will search firstly in our Global scope then built-in scope.
- Non-local X

Slicing lists

```
1 my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
2   0, 1, 2, 3, 4, 5, 6, 7, 8, 9
3   -10, -9, -8, -7, -6, -5, -4, -3, -2, -1
4   #
5
6 # list[start:end:step]
7
8 print my_list[1:9]
9
10 my_list[start:end:step]
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
```

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

[Finished in 0.1s]

```
19 sample_url = 'http://coreyms.com'
20 print sample_url
21
22
23 # Reverse the url
24 # print sample_url[::-1]
25
26 ## Get the top level domain
27 # print sample_url[-4:]
28
29 ## Print the url without the http://
30 # print sample_url[7:]    *
31
32 ## Print the url without the http:// or the top level domain
33 # print sample_url
```

http://coreyms.com
coreyms.com
[Finished in 0.1s]

Comprehensions :

①

```
comprehensions.py
1 nums = [1,2,3,4,5,6,7,8,9,10]
2
3 # I want 'n' for each 'n' in nums
4 # my_list = []
5 # for n in nums:
6 #     my_list.append(n)
7 # print my_list
8
9 my_list = [n for n in nums]
10 print my_list
11
```

③

```
1 nums = [1,2,3,4,5,6,7,8,9,10]
2
3 # I want 'n' for each 'n' in nums if 'n' is even
4 # my_list = []
5 # for n in nums:
6 #     if n%2 == 0:
7 #         my_list.append(n)
8 # print my_list
9
10 my_list = [n for n in nums if n%2 == 0]
11 print my_list
12
13 # Using a filter + lambda
14 # my_list = filter(lambda n: n%2 == 0, nums)
15 # print my_list
```

②

```
2
3 # I want 'n*n' for each 'n' in nums
4 # my_list = []
5 # for n in nums:
6 #     my_list.append(n*n)
7 # print my_list
8
9 my_list = [n*n for n in nums]
10 print my_list
11
12 # Using a map + lambda
13 my_list = map(lambda n: n*n, nums)
14 print my_list
15
```

④

```
1 nums = [1,2,3,4,5,6,7,8,9,10]
2
3 # I want a (letter, num) pair for each letter in 'abcd' and each number in '0123'
4 # my_list = []
5 # for letter in 'abcd':
6 #     for num in range(4):
7 #         my_list.append((letter,num))
8 # print my_list
9
10 my_list = [(letter, num) for letter in 'abcd' for num in range(4)]
11 print my_list
12
```



```

1 nums = [1,2,3,4,5,6,7,8,9,10]
2
3 # Dictionary Comprehensions
4 names = ['Bruce', 'Clark', 'Peter', 'Logan', 'Wade']
5 heros = ['Batman', 'Superman', 'Spiderman', 'Wolverine', 'Deadpool']
6
7 # I want a dict{'name': 'hero'} for each name,hero in zip(names, heros)
8 # my_dict = {}
9 # for name, hero in zip(names, heros):
10#     my_dict[name] = hero
11# print my_dict
12
13 my_dict = {name: hero for name, hero in zip(names, heros) if name != 'Peter'} → if you don't want to add Peter
14 print my_dict
15
16
17 # If name not equal to Peter
18
19 ['Bruce': 'Batman', 'Wade': 'Deadpool', 'Logan': 'Wolverine', 'Peter': 'Spiderman', 'Clark': 'Superman']
[Finished in 0.1s]

```



```

3 # Set Comprehensions
4 nums = [1,1,2,1,3,4,3,4,5,5,6,7,8,7,9,9]
5 # my_set = set()
6 # for n in nums:
7#     my_set.add(n)
8# print my_set
9
10 my_set = {n for n in nums}
11 print my_set
12

```

Note:
set() is like list but
it has only unique value

Set Comprehension

print 'I am CS in CS!', (**per)
I / C (X per)

String Formatting Advanced

```

3
4 tag = 'h1'
5 text = 'This is a headline'
6
7 sentence = '<{0}>{1}</{0}>'.format(tag, text)
8 print(sentence)
9
10

```

```

3
4 person = {'name': 'Jenn', 'age': 23}
5
6 sentence = 'My name is {0[name]} and I am {0[age]} years old.'.format(person)
7 print(sentence)
8

```

```

2 person = {'name': 'Jenn', 'age': 23}
3
4 l = ['Jenn', 23]
5
6 sentence = 'My name is {0[0]} and I am {0[1]} years old.'.format(l)
7 print(sentence)
8

```

```

3
4 sentence = 'My name is {name} and I am {age} years old.'.format(name='Jenn', age=30)
5 print(sentence)
6

```

```

2
3 class Person():
4
5     def __init__(self, name, age):
6         self.name = name
7         self.age = age
8
9 p1 = Person('Jack', 13)
10
11 sentence = 'My name is {0.name} and I am {0.age} years old.'.format(p1)
12 print(sentence)
13

```

```

2
3 person = {'name': 'Jenn', 'age': 23}
4
5 sentence = 'My name is {name} and I am {age} years old.'.format(**person)
6 print(sentence)
7

```

```

2
3 for i in range(1, 11):
4     sentence = 'The value is {}'.format(i)
5     print(sentence)
6

```

→ 1. 14.51 → 2. 14
→ As you can see, when using print 2 digits

OS Module

```
import os
```

`os.getcwd()` → get current dir

Os.Chdir("c:\") → Change dir

os.listdir()

`O.S.mkDir("dir")` → create only dir no subdir create supported

O.S.makedirs("dir1/dir2") → create dir & subdir if needed

os.rmdir ('dir')

`os.remove('dir1/dir2')` → Support to remove subdir, it's danger, be careful with it.

```
os.rename ("oldname.txt", "newname.txt")
```

os.stat("demo.txt") → print file infos

OS.Walk ('path') → will show all dir + subdir + files

Os.environ.get('HOME') → get location of home dir

```
os.path.join(os.environ.get('HOME'), 'test.txt')
```

↳ for solving slash problem.

`os.path.basename('/tmp/text.txt')` → get file name

`os.path.dirname('/tmp/text.txt')` → get dir name

```
os.path.exists('file.txt')
```

```
os.path.isfile('path')
```

```
os.path.isdir('path')
```

`os.path.splitext('/tmp/text.txt')` → split file root & extension

`print(dir(os.path))` → show all function that can be used.

Exercise

```
 2
 3 os.chdir('/Users/coreyschafer/Projects/Demos/Python')
 4
 5 for f in os.listdir():
 6     f_name, f_ext = os.path.splitext(f) → Easier, cleaner
 7
 8     f_title, f_course, f_num = f_name.split('-')
 9
10     f_title = f_title.strip() ↗ remove space
11     f_course = f_course.strip() ↗ strip it
12     f_num = f_num.strip()[1:] ↗ fill(2) ↗
13
14     new_name = '{0}-{1}'.format(f_num, f_title, f_ext)
15     os.rename(f, new_name) ↗
[Finished in 0.1s]
```

Datetime

```
import datetime
```

```
datetime.date(2016, 7, 24)
```

```
datetime.date.today() [year month day]
```

```
datetime.timedelta(days=7) →  
hours=12  
minutes=60
```

datetime.date → weeks, months, years only

datetime.time → hours, minute, second, micro seconds only

datetime.datetime → will give access to all above

```
datetime.time(hours, minutes, seconds, microsecond)
```

```
d = datetime.datetime(year, month, day, hour, minute, second, microsecond)
```

```
print(d.time(), date(), year ...)
```

Same {

- datetime.datetime.today() → return normal
- datetime.datetime.now() → ~ ~ + give us the option to set time
- datetime.datetime.utcnow() →

import pytz
datetime.datetime.today(tz=pytz.UTC)

```
2016-07-19  
[Finished in 0.0s]
```

Files (r, w, e)

`f = open('text.txt', 'r')` r = reading , w = writing , a = append
r+ = read + write , rb, wb = for bytes

`f.name` → print name file
`f.mode` → " mode
`f.read()` → read each line all content
`f.readlines()` → read lines as a list
`f.readline()` → read first line (`end=''`)
`f.tell()` → How much char's is read
`f.seek()` → start reading from beginning of the file, overwrite ok
`f.seek(0)` → will overwrite first character, won't not delete the rest.

```
3 with open('bronx.jpg', 'rb') as rf:  
4     with open('bronx_copy.jpg', 'wb') as wf:  
5         for line in rf:  
6             wf.write(line)  
7  
# File Objects  
#  
3 with open('bronx.jpg', 'rb') as rf:  
4     with open('bronx_copy.jpg', 'wb') as wf:  
5         chunk_size = 4096  
6         rf_chunk = rf.read(chunk_size)  
7         while len(rf_chunk) > 0:  
8             wf.write(rf_chunk)  
9             rf_chunk = rf.read(chunk_size)
```

Random

import random

random.random() → to get value between 0-1 (0.23458...)

random.uniform(1, 10) → " " " " 1-10 (8.345...)

random.randint(1, 10) → " " " " 1-10 (1, 2, 3, 5..)

random.choice(mylist) → grabs random value from mylist

random.choices(mylist, k=10, weights=[25, 50, 25])

show 10 values
Value 1
25% " "
10.000000000000001

random.shuffle(mylist) → print all values, but random places (index)

random.sample(deck, k=5) → print 5 random unique values.

```
9
10 fake_cities = ['Metropolis', 'Eerie', "King's Landing", 'Sunnydale', 'Bedrock', 'South Park', 'Atlantis',
11
12 states = ['AL', 'AK', 'AZ', 'AR', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'ID', 'IL', 'IN', 'IA',
13
14 for num in range(100):
15     first = random.choice(first_names)
16     last = random.choice(last_names)
17
18     phone = f'{random.randint(100, 999)}-555-{random.randint(1000, 9999)}'
19
20     street_num = random.randint(100, 999)
21     street = random.choice(street_names)
22     city = random.choice(fake_cities)
23     state = random.choice(states)
24     zip_code = random.randint(10000, 99999)
25     address = f'{street_num} {street} St., {city} {state} {zip_code}'
26
27     email = first.lower() + last.lower() + '@bogusemail.com'
28
29     print(f'{first} {last}\n{phone}\n{address}\n{email}\n')
```

CSV module

import csv

(2)

```
1 import csv
2
3 with open('names.csv', 'r') as csv_file:
4     csv_reader = csv.reader(csv_file)
5
6     next(csv_reader) # bypass first line
7
8     for line in csv_reader:
9         print(line[2])
10
```

values to csv file

```
john-doe@bogusemail.com
maryjacobs@bogusemail.com
davesmith@bogusemail.com
janestuart@bogusemail.com
tomwright@bogusemail.com
steverobinson@bogusemail.com
nicolejacobs@bogusemail.com
janewright@bogusemail.com
janedoe@bogusemail.com
```

(3)

```
1 import csv
2
3 with open('names.csv', 'r') as csv_file:
4     csv_reader = csv.DictReader(csv_file)
5
6     for line in csv_reader:
7         print(line['email'])
```

```
# with open('new_names.csv', 'w') as new_file:
#     csv_writer = csv.writer(new_file, delimiter='\t')
#
#     for line in csv_reader:
#         csv_writer.writerow(line)
```

```
OrderedDict([('first_name', 'John'), ('last_name', 'Doe'), ('email', 'john-doe@bogusemail.com')])
OrderedDict([('first_name', 'Mary'), ('last_name', 'Smith-Robinson'), ('email', 'maryjacobs@bogusemail.com')])
OrderedDict([('first_name', 'Dave'), ('last_name', 'Smith'), ('email', 'davesmith@bogusemail.com')])
OrderedDict([('first_name', 'Jane'), ('last_name', 'Stuart'), ('email', 'janestuart@bogusemail.com')])
OrderedDict([('first_name', 'Tom'), ('last_name', 'Wright'), ('email', 'tomwright@bogusemail.com')])
OrderedDict([('first_name', 'Steve'), ('last_name', 'Robinson'), ('email', 'steverobinson@bogusemail.com')])
OrderedDict([('first_name', 'Nicole'), ('last_name', 'Jacobs'), ('email', 'nicolejacobs@bogusemail.com')])
OrderedDict([('first_name', 'Jane'), ('last_name', 'Wright'), ('email', 'janewright@bogusemail.com')])
OrderedDict([('first_name', 'Jane'), ('last_name', 'Doe'), ('email', 'janedoe@bogusemail.com')])
```

(4)

```
import csv
with open('names.csv', 'r') as csv_file:
    csv_reader = csv.DictReader(csv_file)
    fieldnames = ['first_name', 'last_name', 'email']
    csv_writer = csv.DictWriter(new_file, fieldnames=fieldnames, delimiter='\t')
    csv_writer.writeheader()
for line in csv_reader:
    csv_writer.writerow(line)
```

(5)

```
import csv
html_output = ''
names = []
with open('patrons.csv', 'r') as data_file:
    csv_data = csv.DictReader(data_file)
    # We don't want first line of bad data
    next(csv_data)
    for line in csv_data:
        if line['FirstName'] == 'No Reward':
            break
        names.append(f'{line["FirstName"]} {line["LastName"]}')
        break
    names.append(f'{line["FirstName"]} {line["LastName"]}')
    html_output += f'

There are currently {len(names)} public contributors. Thank You!

'
    html_output += '\n

\n'
    for name in names:
        html_output += f'<li>{name}</li>\n'
    html_output += '\n</ul>\n'
print(html_output)
```

```
<p>There are currently 30 public contributors. Thank You!</p>
<ul>
<li>John Doe</li>
<li>Dave Smith</li>
<li>Steve Wright</li>
<li>Jane Stuart</li>
<li>Tom Wright</li>
<li>Steve Robinson</li>
<li>Jane Wright</li>
<li>Jane Doe</li>
<li>Kurt Wright</li>
```

Regular expressions

'import re

`print('\tTab')` → \t will make space

`print(r'\tTab')` → because of r, there will not be any space \t

patter = re.compile ('r'abc')

→ compile will put our search in var

.	- Any Character Except New Line
\d	- Digit (0-9)
\D	- Not a Digit (0-9)
\w	- Word Character (a-z, A-Z, 0-9, _)
\W	- Not a Word Character
\s	- Whitespace (space, tab, newline)
\S	- Not Whitespace (space, tab, newline)

\b	- Word Boundary	→ match words with boundaries → = " " without "
\B	- Not a Word Boundary	
^	- Beginning of a String	
\$	- End of a String	

```
re.compile('^\d\d\d.\d\d\d.\d\d\d')
```

[-.3] match dash/dot

↓
match only
dot.

[1-5] match this range

\downarrow \hookrightarrow any character

$\{^{\wedge}a-2A-2\}$ match which is not char

```

29 Mr. Davis
30 Mrs. Robinson
31 Mr. T
32 !!!
33
34 sentence = 'Start a sentence and then bring it to an end'
35
36 pattern = re.compile(r'[89]00[-.]\d\d\d[-.]\d\d\d\d')
37
38 matches = pattern.finditer(text_to_search)
39
40 for match in matches:
41     print(match)
42
43 # with open('data.txt', 'r', encoding='utf-8') as f:
44 #     contents = f.read()
45
46 <_sre.SRE_Match object; span=(194, 206), match='800-555-1234'>
47 <_sre.SRE_Match object; span=(207, 219), match='900-555-1234'>

```

[^b] at match anything
that does not begin
with b followed by at.

[]	- Matches Characters in brackets
[^]	- Matches Characters NOT in brackets
	- Either Or
()	- Group

Quantifiers:

* - 0 or More

+ - 1 or More

? - 0 or One

{3} - Exact Number → match 3 digits $\backslash d\{3\}$

{3,4} - Range of Numbers (Minimum,
Maximum)

```

21 Mr. Schafer
22 Mr Smith
23 Ms Davis
24 Mrs. Robinson
25 Mr. T
26 !!!
27
28 sentence = 'Start a sentence and then bring it
    to an end'
29
30 pattern = re.compile(r'Mr\.\?')
31
32 matches = pattern.finditer(text_to_search)
33
34 for match in matches:
35
36     <_sre.SRE_Match object; span=(221, 224), match='Mr.'>
37     <_sre.SRE_Match object; span=(233, 235), match='Mr.'>
38     <_sre.SRE_Match object; span=(251, 253), match='Mr.'>
39     <_sre.SRE_Match object; span=(265, 268), match='Mr.'>

```

```

21 Mr. Schafer
22 Mr Smith
23 Ms Davis
24 Mrs. Robinson
25 Mr. T
26 !!!
27
28 sentence = 'Start a sentence and then bring it
    to an end'
29
30 pattern = re.compile(r'Mr\.\?\s[A-Z]')
31
32 matches = pattern.finditer(text_to_search)
33
34 for match in matches:
35
36     <_sre.SRE_Match object; span=(221, 226), match='Mr. S'>
37     <_sre.SRE_Match object; span=(233, 237), match='Mr S'>
38     <_sre.SRE_Match object; span=(265, 270), match='Mr. T'>

```

```
1 import re
2
3 sentence = 'Start a sentence and then bring it to an end'
4
5 pattern = re.compile(r'Mr|Ms\s[A-Z]\w*')
6
7 matches = pattern.finditer(text_to_search)
8
9 for match in matches:
10     print(match)
11
<_sre.SRE_Match object; span=(221, 232), match='Mr. Schafer'>
<_sre.SRE_Match object; span=(233, 241), match='Mr Smith'>
```

```
1 import re
2
3 sentence = 'Start a sentence and then bring it to an end'
4
5 pattern = re.compile(r'(Mr|Ms)\s[A-Z]\w*')
6
7 matches = pattern.finditer(text_to_search)
8
9 for match in matches:
10     print(match)
11
<_sre.SRE_Match object; span=(221, 232), match='Mr. Schafer'>
<_sre.SRE_Match object; span=(233, 241), match='Mr Smith'>
<_sre.SRE_Match object; span=(242, 250), match='Ms Davis'>
<_sre.SRE_Match object; span=(251, 264), match='Mrs. Robinson'>
<_sre.SRE_Match object; span=(265, 270), match='Mr. T'>
```

```
1 import re
2
3 emails = ''
4 CoreyMSchafer@gmail.com
5 corey.schafer@university.edu
6 corey-321-schafer@my-work.net
7
8
9 pattern = re.compile(r'[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-].+')
10
11 matches = pattern.finditer(emails)
12
13 for match in matches:
14     print(match)
15
<_sre.SRE_Match object; span=(1, 24), match='CoreyMSchafer@gmail.com'>
<_sre.SRE_Match object; span=(25, 53), match='corey.schafer@university.edu'>
<_sre.SRE_Match object; span=(54, 83), match='corey-321-schafer@my-work.net'>
```

```
1 import re
2
3 urls = ''
4 https://www.google.com
5 http://coreyms.com
6 https://youtube.com
7 https://www.nasa.gov
8
9
10 pattern = re.compile(r'https://(www\.)?(\w+)(\.\w+)')
11
12 matches = pattern.finditer(urls)
13
14 for match in matches:
15     print(match.group(1))
16
www.
None
None
www.
```

```
1 import re
2
3 sentence = 'Start a sentence and then bring it to an end'
4
5 pattern = re.compile(r'Mr|Ms\s[A-Z]\w*')
6
7 matches = pattern.finditer(text_to_search)
8
9 for match in matches:
10     print(match)
11
<_sre.SRE_Match object; span=(221, 232), match='Mr. Schafer'>
<_sre.SRE_Match object; span=(233, 241), match='Mr Smith'>
```

```
1 import re
2
3 emails = ''
4 CoreyMSchafer@gmail.com
5 corey.schafer@university.edu
6 corey-321-schafer@my-work.net
7
8
9 pattern = re.compile(r'[a-zA-Z0-9_.+-]+@[a-zA-Z-]+\.(com|edu|net)')
10
11 matches = pattern.finditer(emails)
12
13 for match in matches:
14     print(match)
15
<_sre.SRE_Match object; span=(1, 24), match='CoreyMSchafer@gmail.com'>
<_sre.SRE_Match object; span=(25, 53), match='corey.schafer@university.edu'>
<_sre.SRE_Match object; span=(54, 83), match='corey-321-schafer@my-work.net'>
```

```
1 import re
2
3 urls = ''
4 https://www.google.com
5 http://coreyms.com
6 https://youtube.com
7 https://www.nasa.gov
8
9
10 pattern = re.compile(r'https://(www\.)?(\w+)(\.\w+)')
11
12 matches = pattern.finditer(urls)
13
14 for match in matches:
15     print(match)
16
<_sre.SRE_Match object; span=(1, 23), match='https://www.google.com'>
<_sre.SRE_Match object; span=(24, 42), match='http://coreyms.com'>
<_sre.SRE_Match object; span=(43, 62), match='https://youtube.com'>
<_sre.SRE_Match object; span=(63, 83), match='https://www.nasa.gov'>
```

```
1 import re
2
3 urls = ''
4 https://www.google.com
5 http://coreyms.com
6 https://youtube.com
7 https://www.nasa.gov
8
9
10 pattern = re.compile(r'https://(www\.)?(\w+)(\.\w+)')
11
12 subbed_urls = pattern.sub(r'\1\2\3', urls)
13
14 print(subbed_urls)
15
google.com
coreyms.com
youtube.com
nasa.gov
```

```
27
28 sentence = 'Start a sentence and then bring it to an end'
29
30 pattern = re.compile(r'Start')↑  
flags .. r.IGNORECASE
31
32 matches = pattern.match(sentence) → Will return the first match, None if no match
33
34 print(matches)
35
36
37
38 <_sre.SRE_Match object; span=(0, 5), match='Start'>
```

Try /except /else /Finally

```
1 exceptions.py
2 test_file.txt
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
848
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
968
969
970
971
972
973
974
975
976
977
978
978
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1156
1157
1158
1158
1159
1160
1161
1162
1163
1164
1165
1165
1166
1167
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1175
1176
1177
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1185
1186
1187
1187
1188
1189
1189
1190
1191
1192
1193
1194
1194
1195
1196
1196
1197
1198
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1215
1216
1217
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1225
1226
1227
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1235
1236
1237
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1245
1246
1247
1247
1248
1249
1249
1250
1251
1252
1253
1254
1255
1255
1256
1257
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1265
1266
1267
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1275
1276
1277
1277
1278
1279
1279
1280
1281
1282
1283
1284
1285
1285
1286
1287
1287
1288
1289
1289
1290
1291
1292
1293
1294
1294
1295
1296
1296
1297
1298
1298
1299
1299
1300
1301
1302
1303
1304
1305
1305
1306
1307
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1315
1316
1317
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1325
1326
1327
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1335
1336
1337
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1345
1346
1347
1347
1348
1349
1349
1350
1351
1352
1353
1354
1355
1355
1356
1357
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1365
1366
1367
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1375
1376
1377
1377
1378
1379
1379
1380
1381
1382
1383
1384
1385
1385
1386
1387
1387
1388
1389
1389
1390
1391
1392
1393
1394
1394
1395
1396
1396
1397
1398
1398
1399
1399
1400
1401
1402
1403
1404
1404
1405
1406
1406
1407
1408
1408
1409
1410
1410
1411
1412
1412
1413
1414
1414
1415
1416
1416
1417
1418
1418
1419
1420
1420
1421
1422
1422
1423
1424
1424
1425
1426
1426
1427
1428
1428
1429
1429
1430
1431
1432
1433
1434
1434
1435
1436
1436
1437
1438
1438
1439
1439
1440
1441
1442
1443
1444
1444
1445
1446
1446
1447
1448
1448
1449
1449
1450
1451
1452
1453
1454
1454
1455
1456
1456
1457
1458
1458
1459
1459
1460
1461
1462
1463
1464
1464
1465
1466
1466
1467
1468
1468
1469
1469
1470
1471
1472
1473
1474
1474
1475
1476
1476
1477
1478
1478
1479
1479
1480
1481
1482
1483
1484
1484
1485
1486
1486
1487
1488
1488
1489
1489
1490
1491
1492
1493
1494
1494
1495
1496
1496
1497
1498
1498
1499
1499
1500
1501
1502
1503
1504
1504
1505
1506
1506
1507
1508
1508
1509
1509
1510
1511
1512
1513
1514
1514
1515
1516
1516
1517
1518
1518
1519
1519
1520
1521
1522
1523
1524
1524
1525
1526
1526
1527
1528
1528
1529
1529
1530
1531
1532
1533
1534
1534
1535
1536
1536
1537
1538
1538
1539
1539
1540
1541
1542
1543
1544
1544
1545
1546
1546
1547
1548
1548
1549
1549
1550
1551
1552
1553
1554
1554
1555
1556
1556
1557
1558
1558
1559
1559
1560
1561
1562
1563
1564
1564
1565
1566
1566
1567
1568
1568
1569
1569
1570
1571
1572
1573
1574
1574
1575
1576
1576
1577
1578
1578
1579
1579
1580
1581
1582
1583
1584
1584
1585
1586
1586
1587
1588
1588
1589
1589
1590
1591
1592
1593
1594
1594
1595
1596
1596
1597
1598
1598
1599
1599
1600
1601
1602
1603
1604
1604
1605
1606
1606
1607
1608
1608
1609
1609
1610
1611
1612
1613
1614
1614
1615
1616
1616
1617
1618
1618
1619
1619
1620
1621
1622
1623
1624
1624
1625
1626
1626
1627
1628
1628
1629
1629
1630
1631
1632
1633
1634
1634
1635
1636
1636
1637
1638
1638
1639
1639
1640
1641
1642
1643
1644
1644
1645
1646
1646
1647
1648
1648
1649
1649
1650
1651
1652
1653
1654
1654
1655
1656
1656
1657
1658
1658
1659
1659
1660
1661
1662
1663
1664
1664
1665
1666
1666
1667
1668
1668
1669
1669
1670
1671
1672
1673
1674
1674
1675
1676
1676
1677
1678
1678
1679
1679
1680
1681
1682
1683
1684
1684
1685
1686
1686
1687
1688
1688
1689
1689
1690
1691
1692
1693
1694
1694
1695
1696
1696
1697
1698
1698
1699
1699
1700
1701
1702
1703
1704
1704
1705
1706
1706
1707
1708
1708
1709
1709
1710
1711
1712
1713
1714
1714
1715
1716
1716
1717
1718
1718
1719
1719
1720
1721
1722
1723
1724
1724
1725
1726
1726
1727
1728
1728
1729
1729
1730
1731
1732
1733
1734
1734
1735
1736
1736
1737
1738
1738
1739
1739
1740
1741
1742
1743
1744
1744
1745
1746
1746
1747
1748
1748
1749
1749
1750
1751
1752
1753
1754
1754
1755
1756
1756
1757
1758
1758
1759
1759
1760
1761
1762
1763
1764
1764
1765
1766
1766
1767
1768
1768
1769
1769
1770
1771
1772
1773
1774
1774
1775
1776
1776
1777
1778
1778
1779
1779
1780
1781
1782
1783
1784
1784
1785
1786
1786
1787
1788
1788
1789
1789
1790
1791
1792
1793
1794
1794
1795
1796
1796
1797
1798
1798
1799
1799
1800
1801
1802
1803
1804
1804
1805
1806
1806
1807
1808
1808
1809
1809
1810
1811
1812
1813
1814
1814
1815
1816
1816
1817
1818
1818
1819
1819
1820
1821
1822
1823
1824
1824
1825
1826
1826
1827
1828
1828
1829
1829
1830
1831
1832
1833
1834
1834
1835
1836
1836
1837
1838
1838
1839
1839
1840
1841
1842
1843
1844
1844
1845
1846
1846
1847
1848
1848
1849
1849
1850
1851
1852
1853
1854
1854
1855
1856
1856
1857
1858
1858
1859
1859
1860
1861
1862
1863
1864
1864
1865
1866
1866
1867
1868
1868
1869
1869
1870
1871
1872
1873
1874
1874
1875
1876
1876
1877
1878
1878
1879
1879
1880
1881
1882
1883
1884
1884
1885
1886
1886
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1894
1895
1896
1896
1897
1898
1898
1899
1899
1900
1901
1902
1903
1904
1904
1905
1906
1906
1907
1908
1908
1909
1909
1910
1911
1912
1913
1914
1914
1915
1916
1916
1917
1918
1918
1919
1919
1920
1921
1922
1923
1924
1924
1925
1926
1926
1927
1928
1928
1929
1929
1930
1931
1932
1933
1934
1934
1935
1936
1936
1937
1938
1938
1939
1939
1940
1941
1942
1943
1944
1944
1945
1946
1946
1947
1948
1948
1949
1949
1950
1951
1952
1953
1954
1954
1955
1956
1956
1957
1958
1958
1959
1959
1960
1961
1962
1963
1964
1964
1965
1966
1966
1967
1968
1968
1969
1969
1970
1971
1972
1973
1974
1974
1975
1976
1976
1977
1978
1978
1979
1979
1980
1981
1982
1983
1984
1984
1985
1986
1986
1987
1988
1988
1989
1989
1990
1991
1992
1993
1994
1994
1995
1996
1996
1997
1998
1998
1999
1999
2000
2001
2002
2003
2004
2004
2005
2006
2006
2007
2008
2008
2009
2009
2010
2011
2012
2013
2014
2014
2015
2016
2016
2017
2018
2018
2019
2019
2020
2021
2022
2023
2024
2024
2025
2026
2026
2027
2028
2028
2029
2029
2030
2031
2032
2033
2034
2034
2035
2036
2036
2037
2038
2038
2039
2039
2040
2041
2042
2043
2044
2044
2045
2046
2046
2047
2048
2048
2049
2049
2050
2051
2052
2053
2054
2054
2055
2056
2056
2057
2058
2058
2059
2059
2060
2061
2062
2063
2064
2064
2065
2066
2066
2067
2068
2068
2069
2069
2070
2071
2072
2073
2074
2074
2075
2076
2076
2077
2078
2078
2079
2079
2080
2081
2082
2083
2084
2084
2085
2086
2086
2087
2088
2088
2089
2089
2090
2091
2092
2093
2094
2094
2095
2096
2096
2097
2098
2098
2099
2099
2100
2101
2102
2103
2104
2104
2105
2106
2106
2107
2108
2108
2109
2109
2110
2111
2112
2113
2114
2114
2115
2116
2116
2117
2118
2118
2119
2119
2120
2121
2122
2123
2124
2124
2125
2126
2126
2127
2128
2128
2129
2129
2130
2131
2132
2133
2134
2134
2135
2136
2136
2137
2138
2138
2139
2139
2140
2141
2142
2143
2144
2144
2145
2146
2146
2147
2148
2148
2149
2149
2150
2151
2152
2153
2154
2154
2155
2156
2156
2157
2158
2158
2159
2159
2160
2161
2162
2163
2164
2164
2165
2166
2166
2167
2168
2168
2169
2169
2170
2171
2172
2173
2174
2174
2175
2176
2176
2177
2178
2178
2179
2179
2180
2181
2182
2183
2184
2184
2185
2186
2186
2187
2188
2188
2189
2189
2190
2191
2192
2193
2194
2194
2195
2
```

Generators

The screenshot shows a Mac OS X desktop with a terminal window open. The terminal window has three tabs: 'square_nums.py', 'people.py', and 'people.py — Generators'. The 'people.py' tab contains the following Python code:

```
1 import mem_profile
2 import random
3 import time
4
5 names = ['John', 'Corey', 'Adam', 'Steve', 'Rick', 'Thomas']
6 majors = ['Math', 'Engineering', 'CompSci', 'Arts', 'Business']
7
8 print 'Memory (Before): {}Mb'.format(mem_profile.memory_usage_resource())
9
10 def people_list(num_people):
11     result = []
12     for i in range(num_people):
13         person = {
14             'id': i,
15             'name': random.choice(names),
16             'major': random.choice(majors)
17         }
18         result.append(person)
19     return result
20
21 def people_generator(num_people):
22     for i in xrange(num_people):
23         person = {
24             'id': i,
25             'name': random.choice(names),
26             'major': random.choice(majors)
27         }
28         yield person
29
30 t1 = time.clock()
31 people = people_list(1000000)
32 t2 = time.clock()
33
34 # t1 = time.clock()
35 # people = people_generator(1000000)
36 # t2 = time.clock()
37
38 print 'Memory (After) : {}Mb'.format(mem_profile.memory_usage_resource())
39 print 'Took {} Seconds'.format(t2-t1)
```

Output from the terminal shows memory usage before and after the execution of the code, along with the execution time.

```
Memory (Before): 15.92578125Mb
Memory (After) : 318.92578125Mb
Took 1.232136 Seconds
[Finished in 1.5s]
```

```
Memory (Before): 15.98046875Mb
Memory (After) : 15.9921875Mb
Took 2e-06 Seconds
[Finished in 0.1s]
```

FOLDERS
Decorators
decorators.py
k.txt
snippets.txt

decoration.py snippets.txt

```
1 # Decorators
2
3
4 def decorator_function(original_function):
5     def wrapper_function():
6         return original_function()
7     return wrapper_function
8
9
10 def display():
11     print('display function ran')
12
13 decorated_display = decorator_function(display)
14
15 decorated_display()
16
```

display function ran
[Finished in 0.0s]

Classes

```
copy x
1 # Python Object-Oriented Programming
2
3
4 class Employee:
5
6     def __init__(self, first, last, pay):
7         self.first = first
8         self.last = last
9         self.pay = pay
10        self.email = first + '.' + last + '@company.com'
11
12    def fullname(self): employee.last or
13        return '{} {}'.format(self.first, self.last)
14
15
16 emp_1 = Employee('Corey', 'Schafer', 50000)
17 emp_2 = Employee('Test', 'User', 60000)
18
19 emp_1.fullname()
20 print(Employee.fullname(emp_1))
21 # print(emp_2.fullname())
22
Corey Schafer
[Finished in 0.0s]
```

```
19 emp_1 = Employee('Corey', 'Schafer', 50000)
20 emp_2 = Employee('Test', 'User', 60000)
21
22 print(emp_1.__dict__)
23
24 # print(Employee.raise_amount)
25 # print(emp_1.raise_amount)
26 # print(emp_2.raise_amount)
27
['first': 'Corey', 'pay': 50000, 'email': 'Corey.Schafer@company.com', 'last': 'Schafer']
[Finished in 0.0s]
```



```
17
18
19 class Developer(Employee):
20     raise_amt = 1.10
21
22     def __init__(self, first, last, pay, prog_lang):
23         super().__init__(first, last, pay)
24         Employee.__init__(self, first, last, pay) } Same
25
26
```

```
008.py      exceptions.py      x
1
2 class Employee:
3
4     raise_amt = 1.04
5
6     def __init__(self, first, last, pay):
7         self.first = first
8         self.last = last
9         self.email = first + '.' + last + '@email.com'
10        self.pay = pay
11
12    def fullname(self):
13        return '{} {}'.format(self.first, self.last)
14
15    def apply_raise(self):
16        self.pay = int(self.pay * self.raise_amt)
17
18
19 class Developer(Employee):
20     raise_amt = 1.10
21
22     def __init__(self, first, last, pay, prog_lang):
23         super().__init__(first, last, pay)
24         self.prog_lang = prog_lang
25
26
27 class Manager(Employee):
28
29     def __init__(self, first, last, pay, employees=None):
30         super().__init__(first, last, pay)
31         if employees is None:
32             self.employees = []
33         else:
34             self.employees = employees
35
36     def add_emp(self, emp):
37         if emp not in self.employees:
38             self.employees.append(emp)
39
40     def remove_emp(self, emp):
41         if emp in self.employees:
42             self.employees.remove(emp)
43
44     def print_emps(self):
45         for emp in self.employees:
46             print('-->'. emp.fullname())
```

Corey.Schafer@email.com

Python

[Finished in 0.0s]

```
1
2 class Employee:
3
4     raise_amt = 1.04
5
6     def __init__(self, first, last, pay):
7         self.first = first
8         self.last = last
9         self.email = first + '.' + last + '@email.com'
10        self.pay = pay
11
12    def fullname(self):
13        return '{} {}'.format(self.first, self.last)
14
15    def apply_raise(self):
16        self.pay = int(self.pay * self.raise_amt)
17
18    def __repr__(self):
19        return "Employee('{}', '{}', {})".format(self.first, self.last, self.pay)
20
21    def __str__(self):
22        return '{} - {}'.format(self.fullname(), self.email)
23
24
25 emp_1 = Employee('Corey', 'Schafer', 50000)
26 emp_2 = Employee('Test', 'Employee', 60000)
27
28 # print(emp_1)
29
30 # print(repr(emp_1))
31 # print(str(emp_1))
32
33 print(emp_1.__repr__())
34 print(emp_1.__str__())
35
```

Employee('Corey', 'Schafer', 50000)
Corey Schafer - Corey.Schafer@email.com
[Finished in 0.0s]

BeautifulSoup

```
Soup = BeautifulSoup(html_file, 'lxml')
```

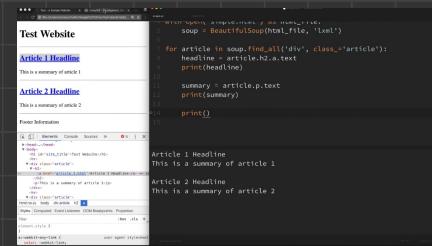
```
match = Soup.title
```

```
= soup.title.text
```

```
= Soup.dev
```

```
= soup.find('dev')
```

```
= soup.find('dev', class_='footer')
```



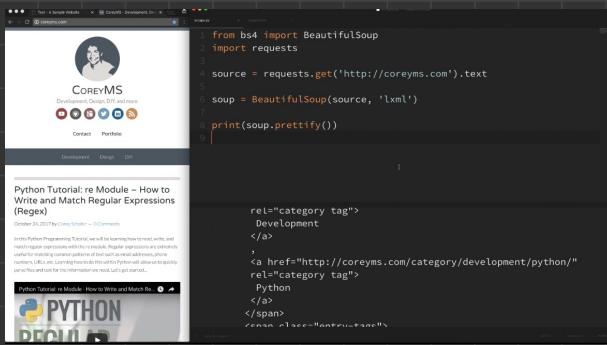
```
from bs4 import BeautifulSoup
import requests

source = requests.get('http://coreyms.com').text

soup = BeautifulSoup(source, 'lxml')

print(soup.prettify())

```



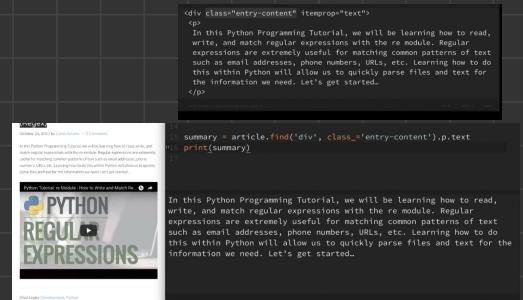
```
from bs4 import BeautifulSoup
import requests

source = requests.get('http://coreyms.com').text

soup = BeautifulSoup(source, 'lxml')

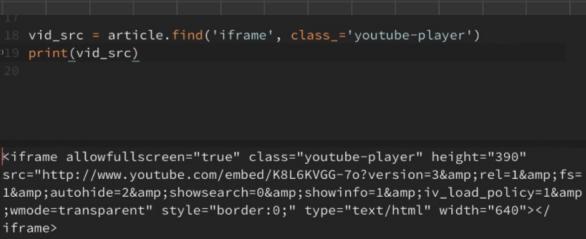
print(soup.prettify())

```



```
<div class="entry-content" itemprop="text">
    ...
</div>

In this Python Programming Tutorial, we will be learning how to read, write, and match regular expressions with the re module. Regular expressions are extremely useful for matching common patterns of text such as email addresses, phone numbers, URLs, etc. Learning how to do this within Python will allow us to quickly parse files and text for the information we need. Let's get started.
</p>
```



```
17
18 vid_src = article.find('iframe', class_='youtube-player')
19 print(vid_src)
20

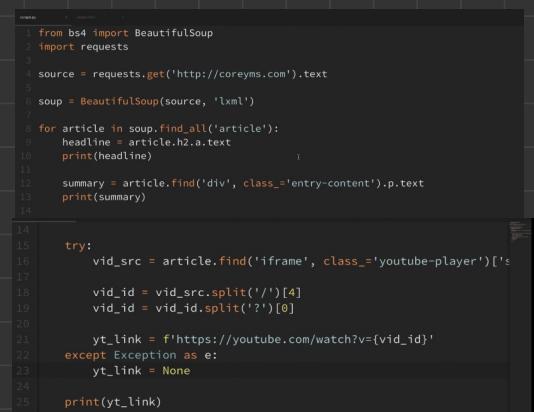
<iframe allowfullscreen="true" class="youtube-player" height="390"
src="https://www.youtube.com/embed/K8L6KVGG-7o?version=3&rel=1&fs=1&autohide=2&showsearch=0&showinfo=1&iv_load_policy=1&mmodo=transparent" style="border:0;" type="text/html" width="640"></
iframe>
```

We want the url id ↗



```
17
18 vid_src = article.find('iframe', class_='youtube-player')['src']
19 # print(vid_src)
20
21 vid_id = vid_src.split('/')[4]
22 vid_id = vid_id.split('?')[0]
23 print(vid_id)
24

K8L6KVGG-7o
```



```
from bs4 import BeautifulSoup
import requests

source = requests.get('http://coreyms.com').text

soup = BeautifulSoup(source, 'lxml')

for article in soup.find_all('article'):
    headline = article.h2.a.text
    print(headline)

    summary = article.find('div', class_='entry-content').p.text
    print(summary)
```

Working with JSON

Simple example

```

1  /** JavaScript Object Notation ***/
2  import json
3
4  people_string = '''
5  {
6    "people": [
7      {
8        "name": "John Smith",
9        "phone": "615-555-7164",
10       "emails": ["johnsmith@bogusemail.com", "john.smith@work-place.com"],
11       "has_license": false
12     },
13     {
14       "name": "Jane Doe",
15       "phone": "560-555-5153",
16       "emails": null,
17       "has_license": true
18   }
19 ]
20 ...
21 ...
22
23 data = json.loads(people_string)
24
25 print(data)
26
{'people': [{'name': 'John Smith', 'phone': '615-555-7164', 'emails': ['johnsmith@bogusemail.com', 'john.smith@work-place.com'], 'has_license': False}, {'name': 'Jane Doe', 'phone': '560-555-5153', 'emails': None, 'has_license': True}]}

```

print json string

```

22
23 data = json.loads(people_string)
24
25 for person in data['people']:
26   print(person['name'])
27
28
John Smith
Jane Doe

```

print only the names

```

22
23 data = json.loads(people_string)
24
25 for person in data['people']:
26   del person['phone']
27
28 new_string = json.dumps(data) # indent=2, sort_keys=True
29
30 print(new_string)
{"people": [{"name": "John Smith", "emails": ["johnsmith@bogusemail.com", "john.smith@work-place.com"], "has_license": false}, {"name": "Jane Doe", "emails": null, "has_license": true}]}

```

Delete users number

```

1  import json
2  from urllib.request import urlopen
3
4  with urlopen("https://finance.yahoo.com/webservice/v1/symbols/allcurrencies/
5   quote?format=json") as response:
6    source = response.read()
7
8  data = json.loads(source)
9
10 print(json.dumps(data, indent=2))
11

```

```

"list": {
"meta": {
  "type": "resource-list",
  "start": 0,
  "count": 188
},
"resources": [
{
  "resource": {
    "classname": "Quote",
    "fields": {
      "name": item['resource']['fields']['name'],
      "price": item['resource']['fields']['price']
    }
  }
}
]
}

```

```

USD/OMR 47.409991
USD/ALL 113.230003
USD/HTG 61.639999
USD/AMD 486.920013
USD/KMF 423.500000
USD/MRO 351.000000
USD/JPY 104.450000
USD/NTD 61.639999
USD/KHR 4031.800000
USD/PHP 50.810001
USD/PRY 0.000000
CYP/E 0.000000

```

19.2.2. Encoders and Decoders

```
class json.JSONDecoder(*, object_hook=None, parse_float=None)
```

Simple JSON decoder.

Performs the following translations in decoding by default:

JSON	Python
object	dict
array	list
string	str
number (int)	int
number (real)	float
true	True
false	False
null	None

```

1  /** JavaScript Object Notation ***/
2  import json
3
4  with open('states.json') as f:
5    data = json.load(f)
6
7  for state in data['states']:
8    del state['area_codes']
9
10 with open('new_states.json', 'w') as f:
11   json.dump(data, f)

```

data that we want to dump

load json from file, delete area code and write results to new file

```

9  # print(json.dumps(data, indent=2))
10 usd_rates = dict()
11
12 for item in data['list']['resources']:
13   name = item['resource']['fields']['name']
14   price = item['resource']['fields']['price']
15   usd_rates[name] = price
16
17 print(50 * float(usd_rates['USD/INR']))
18

```

3240.250000000005

results

