PythonExamples

January 25, 2024

[ ]:
```
print("Hello World!")
```

```
Hello World!
```

[ ]:
```
a = 10
```

[ ]:
```
a
```

[ ]: 10

[ ]:
```
import math
math.pi
```

[ ]: 3.141592653589793

[ ]:
```
type(math.pi)
```

[ ]: float

[ ]:
```
pi = str(math.pi)
```

[ ]:
```
pi
```

[ ]: '3.141592653589793'

[ ]:
```
pi[0:4]
```

[ ]: '3.14'

[ ]:
```
float(pi[0:4])
```

[ ]: 3.14

[ ]:
```
f'The approximate value of pi is {math.pi:.2f}.'
```

[ ]: 'The approximate value of pi is 3.14.'

[ ]:
```
a, b = 1, 2
print (a, b)
```

```
1 2
```

[ ]:
```
temp = a
a = b
b = temp
print (a, b)
```

```
2 1
```

[ ]:
```
a, b = 1, 2
print (a, b)
```

```
1 2
```

[ ]:
```
a = b
b = a
print(a, b)
```

```
2 2
```

[ ]:
```
a, b = 1, 2
print (a, b)
```

```
1 2
```

[ ]:
```
a, b = b, a
```

[ ]:
```
print(a, b)
```

```
2 1
```

[ ]:
```
if a:
  b *= 10
print(b)
```

```
10
```

[ ]:
```
sum = 0
for i in range(10):
  sum += 0.1
print(sum)
```

```
0.9999999999999999
```

[ ]:
```
f'The actual value of 0.1 on this computer is {0.1:.20f}.'
```

[ ]:
```
'The actual value of 0.1 on this computer is 0.10000000000000000555.'
```

[ ]:
```
s = 'UNC Charlotte'
s.find('har')
```

[ ]:
```
5
```

[ ]:
```
s.find('hare')
```

[ ]: -1

[ ]:
```
None
```

[ ]:
```
s = "UNC Charlotte's campus is full of hares."
s.rfind('har')
```

[ ]: 34

[ ]:
```
s.split()
```

[ ]: ['UNC', "Charlotte's", 'campus', 'is', 'full', 'of', 'hares.']

[ ]:
```
s = 'UNC Charlotte's campus is full of hares.'
```

```
  File "<ipython-input-29-abab82ca6718>", line 1
    s = 'UNC Charlotte's campus is full of hares.'
                                                  ^
SyntaxError: unterminated string literal (detected at line 1)
```

[ ]:
```
s
```

[ ]: "UNC Charlotte's campus is full of hares."

[ ]:
```
s = 'charlotte'
s[0:5], s[:5]
```

[ ]: ('charl', 'charl')

[ ]:
```
s[0:5:2]
```

[ ]: 'cal'

[ ]:
```
s[-1], s[-2], s[-2:]
```

[ ]: ('e', 't', 'te')

[ ]:
```
s[::-1]
```

[ ]: 'ettolrahc'

[ ]:
```
s
```

[ ]: 'charlotte'

[ ]:
```
s[0] = 's'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-37-910e3094fc67> in <cell line: 1>()
----> 1 s[0] = 's'

TypeError: 'str' object does not support item assignment
```

[ ]:
```
s, s * 2
```

[ ]:
```
('charlotte', 'charlottecharlotte')
```

[ ]:
```
s + s
```

[ ]:
```
'charlottecharlotte'
```

[ ]:
```
x = list(range(1, 11))
```

[ ]:
```
x
```

[ ]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[ ]:
```
x[::2], x[1::2]
```

[ ]:
```
([1, 3, 5, 7, 9], [2, 4, 6, 8, 10])
```

[ ]:
```
x[2] += 2
x[5] += 3
x
```

[ ]:
```
[1, 2, 5, 4, 5, 9, 7, 8, 9, 10]
```

[ ]:
```
[j * 10 for j in x if j % 2 == 0]
```

[ ]:
```
[20, 40, 80, 100]
```

[ ]:
```
[j for j in x if j % 2 == 0]
```

[ ]:
```
[2, 4, 8, 10]
```

[ ]:
```
x
```

[ ]:
```
[1, 2, 5, 4, 5, 9, 7, 8, 9, 10]
```

[ ]:
```
y = tuple(x)
y
```

[ ]: (1, 2, 5, 4, 5, 9, 7, 8, 9, 10)

[ ]:
```
y[0] = 0
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-50-b8cef91e0169> in <cell line: 1>()
----> 1 y[0] = 0

TypeError: 'tuple' object does not support item assignment
```

[ ]:
```
def prod_sum(a, b):
  return a + b, a * b

y = prod_sum(2, 3)
type(y)
```

[ ]: tuple

[ ]:
```
y
```

[ ]: (5, 6)

[ ]:
```
(5,)
```

[ ]: (5,)

[ ]:
```
a
```

[ ]: 2

[ ]:
```
if a == 0:
  print('nothing')
else:
  if a == 1:
    print('one thing')
  else:
    if a == 2:
      print('two things')
    else:
      if a == 3:
        print('three things')
      else:
        print('many things')
```

```
two things
```

[ ]:
```
if a == 0:
  print('nothing')
elif a == 1:
  print('one thing')
elif a == 2:
  print('two things')
elif a == 3:
  print('three things')
else:
  print('many things')
```

```
two things
```

[ ]:
```
x
```

[ ]:
```
[1, 2, 5, 4, 5, 9, 7, 8, 9, 10]
```

[ ]:
```
for i in range(len(x)):
  print(i, x[i], end = ' ')
```

```
0 1 1 2 2 5 3 4 4 5 5 9 6 7 7 8 8 9 9 10
```

[ ]:
```
def fib_gen():
  a, b = 1, 1
  while True:
    yield a
    a, b = b, a + b

gen = fib_gen()
print(next(gen))
```

```
1
```

[ ]:
```
for i in range(10):
  print(next(gen))
```

```
1
2
3
5
8
13
21
34
55
89
```

[ ]:
```
for i in range(10):
  print(next(gen))
```

```
144
233
377
610
987
1597
2584
4181
6765
10946
```

Write a function `find_sublist(a, b)` that returns `True` if and only if the list `b` appears somewhere in `a`. For example: * `find_sublist([-10, 2, 5, -2, 3], [2, 5])` should return `True`. * `find_sublist([-10, 2, 5, -2, 3], [2, 7])` should return `False`.

[ ]:
```
def find_sublist(a, b):
  for i in range(len(a)):
    # Try to see if b appears starting at position i in a.
    found = True
    for j in range(len(b)):
      if b[j] != a[i + j]:
        found = False
        break
    if found:
      return True
  return False
```

[ ]:
```
a = [-10, 2, 5, -2, 3]
b1 = [2, 5]
b2 = [2, 5, -2]
b3 = [2, 7]

find_sublist(a, b1)
```

[ ]:
```
True
```

[ ]:
```
find_sublist(a, b2)
```

[ ]:
```
True
```

[ ]:
```
find_sublist(a, b3)
```

[ ]:
```
False
```

The `for` loop in Python can have an `else` clause which gets executed if the loop ends normally.

[ ]:
```
def find_sublist(a, b):
  for i in range(len(a)):
    for j in range(len(b)):
```

```
      if b[j] != a[i + j]:
        break
    else:
      return True
  return False
```

[ ]:
```
find_sublist(a, b1), find_sublist(a, b2), find_sublist(a, b3)
```

[ ]:
```
(True, True, False)
```

Write a function `cnodes(tree)` that coutns the number of nodes in a tree that is represented as a list showing the pre-order traversal.

[1]:
```
def cnodes(tree):
  total = 1
  for st in tree[1:]:
    total += cnodes(st)
  return total

tree = ['10', ['7'], ['-1', [5], [6]], [4, [3]]]

cnodes(tree)
```

[1]:
```
7
```

1 Practice problems

- Write a function `remove_duplicates(a)` that takes as input a sorted list and return a list where all duplicates are removed. For example, `remove_duplicates([1, 2, 2, 4, 6, 6, 6, 9, 10, 11, 11, 11, 11, 13, 14, 14])` should return `[1, 2, 4, 6, 9, 10, 11, 13, 14]`.
- Write a function `remove_duplicates(a)` that removes duplicates from a list that is not necessarily sorted. For example, `remove_duplicates([-3, 4, 2, 4, -3, 2])` should return `[-3, 4, 2]`.
- Write a function to find the longest common prefix string amongst an array of strings. For example, `longest-prefix(["flower","flow","flight"])` should return `'fl'`.
- Generate a list of all permutations of the elements in a list. For example, `perm([1, 2, 3])` should output `[[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 1, 3], [3, 1, 2], [3, 2, 1]]`.
- Consider a pre-order tree representation using lists, where a tree T with the value `R` in the root node and subtrees T1, T2, ..., Tn is represented as a list `[R, T1 T2, ..., Tn]`. For example, this tree would be represented as `[2, [7, [2], [20], [6, [5], [11]]], [5, [9, [4]]]]`. Write the functions:
  - `count_nodes(t)` that count the nodes of a tree. For the tree above it should return `10`.

- `count_leaves(t)` that counts the leaves of a tree. For the tree above it should return 5.
- `height(t)` that calculates the height of a tree. The the tree above it should return 3.
- `find(t, x)` that returns `Treu` if and only if t contains the number `x`.

[ ]:
```
import antigravity
```

[ ]:
```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

1.1 Practice problems solutions

[ ]:
```
def longest_prefix(slist):
    prefix = ''
    for i in range(len(slist[0])):
        for s in slist[1:]:
            if i >= len(s):
                return prefix
            if s[i] != slist[0][i]:
                return prefix
        else:
            prefix = prefix + slist[0][i]
    return prefix

longest_prefix(["flower","flow","flight"])
```

[ ]:
```
'fl'
```

[ ]:
```
# Nicer version.
def longest_prefix(slist):
    prefix = ''
    first = slist[0]
    rest = slist[1:]
    for i in range(len(first)):
        for s in rest:
            # Check if the string s is shorter than first.
            if i >= len(s):
                return prefix
            # Check if the characters match.
            if s[i] != first[i]:
                return prefix
        else:
            prefix = prefix + first[i]
    return prefix

longest_prefix(["flower","flow","flight"])
```

[ ]:
```
'fl'
```

[ ]:
```
longest_prefix(["flower","flow"])
```

[ ]:
```
'flow'
```

[ ]:
```

```