```
a = 1
```

```
a
```

```
1
```

```
print(a)
```

```
1
```

```
print("Hello world")
```

```
Hello world
```

```
# Factorial function: iterative
def fact(n):
  result = 1
  while n > 1:
    result = result ∗ n
    n = n − 1

  return result
```

```
fact(100)
```

```
93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185
```

```
# Factorial function: iterative
def fact(n):
  if n == 0:
    return 1
  return n ∗ fact(n − 1)

  return result
```

```
fact(100)
```

```
93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185
```

```
type(fact)
```

```
function
```

```
a = 0.1
type(a)
```

```
float
```

```
a = 0.1
sum = 0
for _ in range(10):
  sum = sum + a
print(sum)
```

```
0.9999999999999999
```

```
type(a)
```

```
float
```

```
print(a)
```

```
0.1
```

```
# Formatted string
f"a is really {a : .20f}"
```

```
'a is really  0.10000000000000000555'
```

```
list(range(10))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
list(range(1, 10))
```

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

```
list(range(1, 10, 2))
```

```
[1, 3, 5, 7, 9]
```

```
l = [1, 2, 3, 4, 5]
type(l)
```

```
list
```

```
l = list(range(20))
```

```
l
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

```
l = l[2:11]
print(l)
```

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

```
l = l[2:11:2]
print(l)
```

```
[4, 6, 8, 10]
```

```
l[1:len(l)]
```

```
[6, 8, 10]
```

```
l[1:]
```

```
[6, 8, 10]
```

```
l
```

```
[4, 6, 8, 10]
```

```
l[:3]
```

```
[4, 6, 8]
```

```
l[:]
```

```
[4, 6, 8, 10]
```

```
lnew = l[::-1]
print(lnew)
```

```
[10, 8, 6, 4]
```

```
lnew[1] = 10
print(lnew)
```

```
[10, 10, 6, 4]
```

```
print(l)
```

```
[4, 6, 8, 10]
```

```
l[3] = 8.5
print(l)
```

```
[4, 6, 8, 8.5]
```

```
l[0] = 'charlotte'
print(l)
```

```
['charlotte', 6, 8, 8.5]
```

```
t = (1, 3, 5, 5)
type(t)
```

```
tuple
```

```
# Python tuples are immutable
t[0] = 0
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-57-51595a4035f3> in <cell line: 1>()
----> 1 t[0] = 0

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

```
a = 1, 2, 3
print(a)
```

```
(1, 2, 3)
```

```
# This does tuple assignment (a, b) = (1, 2)
a, b = 1, 2
```

```
print (a, b)
```

```
1 2
```

```
temp = a
a = b
b = temp
print(a, b)
```

```
2 1
```

+ Code + Text

```
a, b = b, a
print(a, b)
```

```
1 2
```

```
math.pi
```

```
3.141592653589793
```

```
spi = str(math.pi)
spi
```

```
'3.141592653589793'
```

```
float(spi)
```

```
3.141592653589793
```

```
int("2351")
```

```
2351
```

```
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'
```

```
# Python strings are immutable
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
```

```
# Dictionaries
d = {'john': 23, 'alex': 25, 'bill': 99}
print(type(d))
print(d)
```

```
<class 'dict'>
{'john': 23, 'alex': 25, 'bill': 99}
```

```
'alex' in d
```

```
True
```

```
'mary' in d
```

```
False
```

```
d['mary'] = 30
d
```

```
{'john': 23, 'alex': 25, 'bill': 99, 'mary': 30}
```

```
d['john'] = 35
d
```

```
{'john': 35, 'alex': 25, 'bill': 99, 'mary': 30}
```

```
d['harry']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-17-62aaa5814145> in <cell line: 1>()
----> 1 d['harry']

KeyError: 'harry'
```

```
# Two solutions for initializing value (if key non-existent) or updating value (if key exists in dictionary).

key = 'harry'
# Solution 1
if key in d:
  d[key] += 10
else:
  d[key] = 10

key = 'barry'
#Solution 2
d[key] = d.get(key, 0) + 10

print(d)
```

```
{'john': 35, 'alex': 25, 'bill': 99, 'mary': 30, 'harry': 10, 'barry': 10}
```

```
del d['harry']
del d['barry']
```

```
# Note right-branching effect of if-else, and one way of including quotes in a Python string.
a = 5
if a == 0:
  print('nada')
else:
  if a == 1:
    print('uno')
  else:
    if a == 2:
      print('dos')
    else:
      if a == 3:
        print('tres')
      else:
        print("I'm tired")
```

```
I'm tired
```

```
# elif statements eliminate right-branching.
a = 5
if a == 0:
  print('nada')
elif a == 1:
  print('uno')
elif a == 2:
  print('dos')
elif a == 3:
  print('tres')
else:
  print("I'm tired")
```

```
I'm tired
```

```
# If x belongs to a, return True (else with for).
def search(a, x):
  for e in a:
    if e == x:
      return True
  else:
    return False
```

```
search([1, 2, 3, 5], 4)
```

```
False
```

```
l1 = [1, 2, 3]
l1.append(4)
print(l1)
```

```
[1, 2, 3, 4]
```

```
# Fibonnacci numbers 1, 1, 2, 3, 5, 8, 13, 21, ...
# Function that generates a list with the first n Fibonnacci numbers.
def fibo1(n):
  if n == 1:
    return [1]

  if n == 2:
    return [1, 1]

  a, b = 1, 1
  result = [a, b]
  for _ in range(n-2):
    a, b = b, a + b
    result.append(b)

  return result

print(fibo1(30))
```

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 1
```

```
# Fibonnacci numbers 1, 1, 2, 3, 5, 8, 13, 21, ...
# Function that generates the nth Fibonnacci numbers.
def fibo1(n):
  a, b = 0, 1
  for _ in range(n):
    a, b = b, a + b

  return a

fibo1(100)
```

```
354224848179261915075
```

```
# Fibonnacci numbers 1, 1, 2, 3, 5, 8, 13, 21, ...
# Function that generates the nth Fibonnaci number
def fibo2(n):
  if n == 1:
    return 1

  if n == 2:
    return 1

  return fibo2(n-1) + fibo2(n-2)

fibo2(100)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-103-ca69cc326b77> in <cell line: 12>()
     10   return fibo2(n-1) + fibo2(n-2)
     11
---> 12 fibo2(100)

    83 frames
<ipython-input-103-ca69cc326b77> in fibo2(n)
      5     return 1
      6
----> 7   if n == 2:
      8     return 1
      9

KeyboardInterrupt:
```

```
def fibo():
  a, b, = 1, 1
  while True:
    yield a
    a, b = b, a + b
```

```
gen = fibo()
```

```
next(gen)
```

```
1
```

```
next(gen)
```

```
1
```

```
next(gen)
```

```
2
```

```
next(gen)
```

```
3
```

```
for _ in range(20):
  print(next(gen), end = ' ')
```

```
5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368
```

```
# Use `cmath` if you need to generate complex numbers.
import math

def quad_sol(a, b, c):
  """
    This function calculates the solutions of a quadratic equation ...
  """
  term = math.sqrt(b * b - 4 * a * c)
  sol1 = (-b + term) / (2 * a)
  sol2 = (-b - term) / (2 * a)

  return sol1, sol2
```

```
quad_sol(1, -4, 3)
```

```
((3+0j), (1+0j))
```

```
quad_sol(1, 1, 1)
```

```
((-0.5+0.8660254037844386j), (-0.5-0.8660254037844386j))
```

```
def seq1():
  a = [1, 2, 4, 7, 11, 16, 22, 29, 37, 46, 56, 67, 79, 94]
  while True:
    for e in a:
      yield e
```

```
gen1 = seq1()
for i in range(30):
  print(next(gen1), end = ' ')
```

```
1 2 4 7 11 16 22 29 37 46 56 67 79 94 1 2 4 7 11 16 22 29 37 46 56 67 79 94 1 2
```

```
def seq2():
  a, d = 1, 1
  while True:
    yield a
    a, d = a + d, d + 1
```

```
gen2 = seq2()
for i in range(30):
  print(next(gen2), end = ' ')
```

```
1 2 4 7 11 16 22 29 37 46 56 67 79 92 106 121 137 154 172 191 211 232 254 277 301 326 352 379 407 436
```

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)
```

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], [10], [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`.