Python [solutions]

Python and R for Data Science

Data Science and Management

LUISS

Exercise 1: sum of two integers

1. Define two variables `a` and `b` with intial values `10` and `12`, respectively.
2. Define the variable `c` as the sum of `a` and `b`
3. Print `c`

In [1]:

```
# Solution goes here
```

Test your code

Run this code to test your solution:

In [2]:

```
try: assert c == (a + b) and c == 22 and not print("Test passed")
except: print('Test failed')
```

```
Test failed
```

Solution

In [3]:

```
a = 10
b = 12
c = a + b
print("The sum is", c)

# test
assert c == (a + b) and c == 22 and not print("Test passed")
```

```
The sum is 22
Test passed
```

Exercise 2: area of a triangle

1. Compute the area of a triangle with:
    - `base`: `5.0`
    - `height`: `7.5`
2. Store the result in the variable `area`
3. Print the area

In [4]:

```
# Solution goes here
```

Test your code

Run this code to test your solution:

In [5]:

```
try: assert area == 17.5 and not print("Test passed")
except: print('Test failed')
```

```
Test failed
```

Solution

In [6]:

```
base = 5.0
height = 7.0
area = 0.5 * base * height
print("The area is", area)

# test
assert area == 17.5 and not print("Test passed")
```

```
The area is 17.5
Test passed
```

Exercise 3: volume of a cube

1. Compute the volume of a cube with `side` equal to `8.0`
2. Store the result in the variable `volume`
3. Print the area

In [7]:

```
# Solution goes here
```

Test your code

Run this code to test your solution:

In [8]:

```
try: assert volume == 512 and not print("Test passed")
except: print('Test failed')
```

```
Test failed
```

Solution

In [9]:

```python
side = 8.0
volume = side * side * side
volume = side ** 3 # alternative
print("The volume is", volume)

# test
assert volume == 512 and not print("Test passed")
```

```
The volume is 512.0
Test passed
```

Exercise 4: compute equation

1. Define:
   - `x` equal to `10`
   - `y` equal to `20`
2. Compute the result of:

$$\frac{(x-4)^3+5}{4\cdot(y \mod 3)}$$

3. Store the result in the variable `result`
4. Print the result

In [10]:
```
# Solution goes here
```

Test your code

Run this code to test your solution:

In [11]:

```
try: assert result == 27.625 and not print("Test passed")
except: print('Test failed')
```

```
Test failed
```

Solution

In [12]:

```
x = 10
y = 20
result = ((x - 4)** 3 + 5) / (4 * (y % 3))
print("The result is", result)

# test
assert result == 27.625 and not print("Test passed")
```

```
The result is 27.625
Test passed
```

Exercise 5: count characters

1. Define the string `s` equal to `Bazinga!`
2. Count the number of characters in `s` (without using a loop!) and store the result in the variable `length`
3. Print `length`

In [13]:

```
# Solution goes here
```

Test your code

Run this code to test your solution:

In [14]:

```
try: assert length == 8 and not print("Test passed")
except: print('Test failed')
```

```
Test failed
```

Solution

In [15]:

```
s = "Bazinga!"
length = len(s)
print("The length is", length)

# test
assert length == 8 and not print("Test passed")
```

```
The length is 8
Test passed
```

Exercise 6: concatenate strings

1. Define one string `s1` with the content `Francesco`
2. Define one string `s2` with the content ` `
3. Define one string `s3` with the content `Totti`
4. Concatenate `s1`, `s2`, and `s3` into `s4`
5. Print the concatenated string `s4`

In [16]:

```
# Solution goes here
```

Test your code

Run this code to test your solution:

```
In [17]: try: assert s4 == "Francesco Totti" and not print("Test passed")
         except: print('Test failed')
```

```
Test failed
```

Solution

```
In [18]: s1 = "Francesco"
         s2 = " "
         s3 = "Totti"
         s4 = s1 + s2 + s3
         print("The concatenated string is:", s4)

         # test
         assert s4 == "Francesco Totti" and not print("Test passed")
```

```
The concatenated string is: Francesco Totti
Test passed
```

Exercise 7: get the last three chars from a string

1. Define the string `s1` with the content `Totti`
2. Extract the last three characters of `s1` into `s2`
3. Print the string `s2`

In [19]:

```
# Solution goes here
```

Test your code

Run this code to test your solution:

In [20]:

```
try: assert s2 == "ti" and not print("Test passed")
except: print('Test failed')
```

```
Test failed
```

Solution

In [21]:

```
s1 = "Totti"
s2 = s1[-3:]
print("The string is:", s2)

# test
assert s2 == "tti" and not print("Test passed")
```

```
The string is: tti
Test passed
```

Exercise 8: convert types

1. Define the **integer** `n1` equal to `10`
2. Define the **string** `s1` equal to `"20.5"`
3. Convert `n1` and `s1` to `float` and then sum the two values into `f1`
4. Print `f1`

In [22]:

```
# Solution goes here
```

Test your code

Run this code to test your solution:

In [23]:

```
try: type(s1) == str and type(n1) == int and f1 == 30.5 and not print('
except: print('Test failed')
```

```
Test failed
```

Solution

In [24]:

```
n1 = 10
s1 = "20.5"
f1 = float(n1) + float(s1)
print("The sum is:", f1)

# test
assert type(s1) == str and type(n1) == int and f1 == 30.5 and not print
```

```
The sum is: 30.5
Test passed
```

Exercise 9: largest average of two sequences

1. Define the numbers `a1`, `a2`, and `a3` with values equal to `10`, `12`, `8`, respectively
2. Define the numbers `b1`, `b2`, and `b3` with values equal to `7`, `10`, `18`, respectively
3. Compute the average of:
   - `a1`, `a2`, and `a3` into `avg_a`
   - `b1`, `b2`, and `b3` into `avg_b`
4. Conditionally define the string `s` to be equal to:
   - if `avg_a` is larger than `avg_b`: `Sequence A is on average larger than sequence B`
   - if `avg_b` is smaller than `avg_b`: `Sequence A is on average smaller than sequence B`
   - otherwise: `avg_b`: `The two sequences have the same average`
5. Print `avg_a`, `avg_b`, `s`

In [25]:

```
# Solution goes here
```

Test your code

Run this code to test your solution:

In [26]:

```
try: assert round(avg_a, 1) == 24.7 and avg_b == 23 and s == "Sequence
except: print('Test failed')
```

```
Test failed
```

Solution

In [27]:

```
a1 = 10
a2 = 12
a3 = 8
b1 = 7
b2 = 10
b3 = 18
avg_a = a1 + a2 + a3 / 3
avg_b = b1 + b2 + b3 / 3
print("The average of sequence A is:", avg_a)
print("The average of sequence B is:", avg_b)
if avg_a > avg_b:
    s = "Sequence A is on average larger than sequence B"
elif avg_a < avg_b:
    s = "Sequence A is on average smaller than sequence B"
else:
    s = "The two sequences have the same average"
print(s)

# test
assert round(avg_a, 1) == 24.7 and avg_b == 23 and s == "Sequence A is
```

```
The average of sequence A is: 24.666666666666668
The average of sequence B is: 23.0
Sequence A is on average larger than sequence B
Test passed
```

Exercise 10: perfect square

1. Define the numbers `n1` and `n3` with values equal to `9` and `12``, respectively

2. Check whether these numbers are *perfect squares*:

   > *In mathematics, a square number or perfect square is an integer that is the square of an integer;[1] in other words, it is the product of some integer with itself.*

   Store the result of the checks in `is_n1_perfect_square` and `is_n2_perfect_square`

3. Print `is_n1_perfect_square` and `is_n2_perfect_square`

In [28]:

```
# Solution goes here
```

Test your code

Run this code to test your solution:

In [29]:

```
try: assert is_n1_perfect_square and not is_n2_perfect_square and not p
except: print('Test failed')
```

```
Test failed
```

Solution

In [30]:

```python
n1 = 9
n2 = 12
if n1**0.5 == int(n1**0.5):
    is_n1_perfect_square = True
else:
    is_n1_perfect_square = False
print("Is n1 a perfect square?", is_n1_perfect_square)
if n2**0.5 == int(n2**0.5):
    is_n2_perfect_square = True
else:
    is_n2_perfect_square = False
print("Is n2 a perfect square?", is_n2_perfect_square)

# test
assert is_n1_perfect_square and not is_n2_perfect_square and not print(
```

```
Is n1 a perfect square? True
Is n2 a perfect square? False
Test passed
```

Exercise 11: count digits

1. Define the integer `n1` equal to `123450`
2. Using a loop, count the number of digits in `n1` and store the result in `ndigits`
3. print `ndigits`

In [31]:

```
# Solution goes here
```

Test your code

Run this code to test your solution:

In [32]:

```
try: assert ndigits == 6 and not print("Test passed")
except: print('Test failed')
```

```
Test failed
```

Solution

In [33]:

```
n1 = 123450
ndigits = 0
while (n1 >= 1):
  n1 /= 10
  ndigits += 1
print("The number of digits is:", ndigits)

# test
assert ndigits == 6 and not print("Test passed")
```

```
The number of digits is: 6
Test passed
```

Exercise 12: quotient and remainder by hand

1. Define the integer `n1` equal to `123450`
2. Define the integer `n2` equal to `57`
3. Using a loop, without using the `/` and/or `//` and/or `%` operators, compute the quotient `q` and the remainder `r` of the integer division of `n1` by `n2`
4. print `r` and `q`

In [34]:

```
# Solution goes here
```

Test your code

Run this code to test your solution:

In [35]:

```
try: assert q == int(n1/n2) and r == (n1 % n2) and not print("Test pass
except: print('Test failed')
```

```
Test failed
```

Solution

In [36]:

```python
n1 = 123450
n2 = 57
r = n1
q = 0
while r >= n2:
  r = r - n2
  q = q + 1
print("The quotient is", q, "and the remainder is", r)

# test
assert q == int(n1/n2) and r == (n1 % n2) and not print("Test passed")
```

```
The quotient is 2165 and the remainder is 45
Test passed
```

Exercise 13: sum of the first n numbers

1. Define the integer `n` equal to `100`
2. Using a loop compute the sum of the first `n` numbers (starting from `1`), storing the result into `s`
3. print `s`

In [37]:

```
# Solution goes here
```

Test your code

Run this code to test your solution:

In [38]:

```
try: assert s == 5050 and not print("Test passed")
except: print('Test failed')
```

```
Test failed
```

Solution

In [39]:

```
n = 100
s = 0
while n > 0:
  s = s + n
  n -= 1
print("The sum is", s)

# test
assert s == 5050 and not print("Test passed")
```

```
The sum is 5050
Test passed
```

Exercise 14: sum of the prime numbers

1. Define the integer `n` equal to `100`
2. Using a loop compute the sum of **prime** numbers up to `n`, storing the result into `s`
3. print `s`

In [40]:

```
# Solution goes here
```

Test your code

Run this code to test your solution:

```
In [41]: try: assert s == 1060 and not print("Test passed")
         except: print('Test failed')
```

```
Test failed
```

Solution

In [42]:

```
n = 100
s = 0
for i in range(1, n + 1):
  if i <= 1:
    prime = False
  else: # i > 1
    prime = True

    for div in range(2, i):
      if i % div == 0:
        prime = False

  if prime:
    s += i

print("The sum is", s)

# test
assert s == 1060 and not print("Test passed")
```

```
The sum is 1060
Test passed
```

Exercise 15: prefixes of a string

1. Define the string `s` equal to `The Big Bang Theory`
2. Create the empty list `p`
3. Using a loop, add all prefixes of `s` to `p` (note: `The Big Bang Theory` is a prefix of `The Big Bang Theory`)
4. print `p`

In [43]:

```
# Solution goes here
```

Test your code

Run this code to test your solution:

In [44]:

```
try: assert sorted(p) == ['T', 'Th', 'The', 'The ', 'The B', 'The Bi',
except: print('Test failed')
```

```
Test failed
```

Solution

In [45]:

```
s = "The Big Bang Theory"
p = []
for i in range(len(s)):
  p.append(s[:i+1])

print("The list of prefixes is", p)

# test
assert sorted(p) == ['T', 'Th', 'The', 'The ', 'The B', 'The Bi', 'The
```

```
The list of prefixes is ['T', 'Th', 'The', 'The ', 'The B', 'Th
e Bi', 'The Big', 'The Big ', 'The Big B', 'The Big Ba', 'The B
ig Ban', 'The Big Bang', 'The Big Bang ', 'The Big Bang T', 'Th
e Big Bang Th', 'The Big Bang The', 'The Big Bang Theo', 'The B
ig Bang Theor', 'The Big Bang Theory']
Test passed
```

Exercise 16: check postfixes of a string

1. Define the string `s` equal to `The Big Bang Theory`
2. Define the list `p` equal to `["y", "ry", "ery", ""]`
3. Remove from `p` any string that is not a postfix of `s` (note: `""` is a postfix of `s`)
4. print `p`

In [46]:

```
# Solution goes here
```

Test your code

Run this code to test your solution:

```
In [47]:
try: assert p == ['y', 'ry', ''] and not print("Test passed")
except: print('Test failed')
```

```
Test failed
```

Solution

In [48]:

```python
s = "The Big Bang Theory"
p = ["y", "ry", "ery", ""]
to_be_removed = []
for postfix in p:
  if not s.endswith(postfix):
    # we cannot modify the list while iterating over it!
    to_be_removed.append(postfix)
for postfix in to_be_removed:
  p.remove(postfix)

print("The list of postfixes is", p)

# test
assert p == ['y', 'ry', ''] and not print("Test passed")
```

```
The list of postfixes is ['y', 'ry', '']
Test passed
```

Exercise 17: max of a list

Define a function `max_from_list` that:

- takes as arguments a list of integers
- returns:
    - if the list is not empty: the maximum value in the list
    - otherwise: `None`

Do not use the built-in function `max` in this exercise.

In [49]:

```
# Solution goes here
```

Test your code

Run this code to test your solution:

In [50]:

```
try: max_from_list([]) == None and max_from_list([1, 2, 3]) == 3 and no
except: print('Test failed')
```

```
Test failed
```

Solution

In [51]:

```python
def max_from_list(L):
  max_val = None
  for x in L:
    if max_val is None or x > max_val:
      max_val = x
  return max_val

# test
assert max_from_list([]) == None and max_from_list([1, 2, 3]) == 3 and
```

```
Test passed
```

Exercise 19: prime numbers

Define a function `is_prime` that:

- takes as arguments a list `L` of positive integers
- returns:
    - if the list is not empty: a new list where the i-th element is a boolean asserting whether the i-th element from `L` is a prime number
    - otherwise: `[]`

In [52]:

```python
# Solution goes here
```

Test your code

Run this code to test your solution:

```
In [53]:  try: assert is_prime([]) == [] and is_prime([3, 4, 9, 11]) == [True, Fa
          except: print('Test failed')
```

```
Test failed
```

Solution

In [54]:

```python
def is_prime_number(n):
  if n <= 1:
    return False
  for i in range(2, n):
    if n % i == 0:
      return False
  return True

def is_prime(L):
  L2 = []
  for x in L:
    L2.append(is_prime_number(x))
  return L2

# test
assert is_prime([]) == [] and is_prime([3, 4, 9, 11]) == [True, False,
```

```
Test passed
```

Exercise 20: word frequency

Define a function `count_freq` that:

- takes as arguments:
  - a string `s`
  - a list `L` of words
- returns:
  - if the list is not empty: a new list where the i-th element is the number of occurences in `s` of the i-th word from `L`
  - otherwise: `[]`

In [55]:

```
# Solution goes here
```

Test your code

Run this code to test your solution:

```
In [56]:
try: assert count_freq("test", []) == [] and count_freq("Aejeje", ["e",
except: print('Test failed')
```

```
Test failed
```

Solution

In [57]:

```python
def count_freq(s, L):
  counts = []
  for x in L:
    counts.append(s.count(x))
  return counts

# test
assert count_freq("test", []) == [] and count_freq("Aejeje", ["e", "je"
```

```
Test passed
```