The following exercises are related to the Python programming language [1].

1. Write a function `sumultiply` that takes two integer arguments and returns their product. The function should not use the * or / operators. For example:

```
> sumultiply(11, 13)
143
> sumultiply(5, 123)
615
```

Solution:

```
# Ian McLoughlin, 2018-03-14
# https://github.com/ianmcloughlin/problems-python-fundamentals

def sumultiply(x, y):
  # Create a variable that will become the answer.
  total = 0
  # Loop over y, adding x to the total.
  for i in range(y):
    total = total + x
  return total

# Tests from question.
print(sumultiply(11, 13))
print(sumultiply(5, 123))
```

2. Write a function `ispalindrome` that takes a string and returns `True` if the string is a palindrome and `False` otherwise. For example:

```
> ispalindrome("radar")
True
> ispalindrome("radars")
False
```

Solution:

```
# Ian McLoughlin, 2018-03-14
# https://github.com/ianmcloughlin/problems-python-fundamentals

def ispalindrome(s):
  # Create a variable that will become the answer.
  ans = True
  # Loop over the length of the string
```

```
  for i in range(len(s)):
    if s[i] != s[len(s) - i - 1]:
      ans = False
  return ans

# Tests from question.
print(ispalindrome("radar"))
print(ispalindrome("radars"))
```

3. Write a function `simpleinterest` that, for a loan with simple interest, takes a principal amount, an interest rate, and a number of periods, and returns the total amount repaid.

```
> simpleinterest(1000, 3, 5)
1150.0
> simpleinterest(1000, 7, 10)
1700.0
```

Solution:

```
# Ian McLoughlin, 2018-03-14
# https://github.com/ianmcloughlin/problems-python-fundamentals

def simpleinterest(p, r, n):
  # Calculate the interest for one period.
  i = p * (0.01 * r)
  # Calculate the interest for all periods.
  t = i * n
  # Return the total interest plus principal, rounded.
  return round(p + t, 2)

# Tests from question.
print(simpleinterest(1000, 3, 5))
print(simpleinterest(1000, 7, 10))
```

4. Write a function `compoundinterest` that, for a loan with compound interest, takes a principal amount, an interest rate, and a number of periods, and returns the total amount repaid.

```
> compoundinterest(1000, 3, 5)
1159.27
> compoundinterest(1000, 7, 10)
1967.15
```

Solution:

```
# Ian McLoughlin, 2018-03-14
# https://github.com/ianmcloughlin/problems-python-fundamentals

def compoundinterest(p, r, n):
  # Loop over the periods.
  for i in range(n):
    # Increase the principal.
    p = p + (p * (0.01 * r))
  # Return the final principal, rounded to nearest cent.
  return round(p, 2)

# Tests from question.
print(compoundinterest(1000, 3, 5))
print(compoundinterest(1000, 7, 10))
```

5. Write a function `newtonsroot` that takes a number x and returns its square root correct to six decimal places as calculated by Newton's method. Newton's method is to make an initial (random) guess r_0 at the square root, and to repeatedly improve it as follows:

$$r_{i+1} = r_i - \frac{r_i^2 - x}{2r_i}$$

For example:

```
> newtonsroot(100)
10.0
> newtonsroot(144)
12.0
```

Solution:

```
# Ian McLoughlin, 2018-03-14
# https://github.com/ianmcloughlin/problems-python-fundamentals

def newtonsroot(x):
  # Set the initial guess to anything. Try half of x.
  z = x / 2.0
  # Set the next guess using the formula.
  n = z - ((z**2 - x)/(2 * z))
  # Keep looping until the difference between the current guess
  # and the next guess is less than 0.000001.
  while abs(z - n) >= 0.0000001:
    z = n
```

```
    n = z - ((z**2 - x)/(2 * z))
  return n

# Tests from question.
print(newtonsroot(100))
print(newtonsroot(144))
```

6. Write a function `pitondecs` that takes an integer n and returns π correct to n decimal places. For example:

```
> pitondecs(2)
3.14
> pitondecs(6)
3.141593
```

Solution:

```
# Ian McLoughlin, 2018-03-14
# https://github.com/ianmcloughlin/problems-python-fundamentals

def pitondecs(n):
  # See https://www.wikihow.com/Calculate-Pi for method
  # Set pi to 0.
  pi = 0.0
  # Set i to 0.
  i = 0
  # Calculate the next approximation of pi.
  ap = pi + (8.0 / (((4.0 * i) + 1) * ((4.0 * i) + 3.0)))
  # Calculate the minimum change in guesses.
  d = 1.0 / 10**(n+1)
  # Keep looping until the difference between the current guess
  # and the next guess is less than 0.000001.
  # Warning: this doesn't converge quick enough.
  while abs(pi - ap) >= d:
    # Increase i by 1.
    i = i + 1
    # Set pi to the next approximation.
    pi = ap
    # Calculate the next approximation.
    ap = pi + (8.0 / (((4.0 * i) + 1) * ((4.0 * i) + 3.0)))
  # Round pi to n decimal places and return.
  return round(pi, n)

# Tests from question.
```

```
print(pitondecs(2))
print(pitondecs(6))
```

7. Write a function `etondecs` that takes an integer n and returns ϵ correct to n decimal places. For example:

```
> etondecs(2)
2.72
> etondecs(6)
2.718282
```

Solution:

```
True
```

8. Write a function `caesar` that takes a string and an integer n and returns the string with each letter shifted n places in the alphabet. For example:

```
> caesar('abcd', 3)
'defg'
> caesar('Hello, world!', 2)
'Jgnnq, yqtnf!'
```

Solution:

```
# Ian McLoughlin, 2018-03-14
# https://github.com/ianmcloughlin/problems-python-fundamentals

def caesar(s, n):
  # Turn the string into a list.
  s = list(s)
  # Loop through the string a character at a time.
  for i in range(len(s)):
    # Check if the character is a lowercase letter.
    if ord('a') <= ord(s[i]) <= ord('z'):
      # Calculate its new ord number if so.
      newcord = ord(s[i]) + n
      # If the new character has gone beyond 'z'.
      if newcord > ord('z'):
        # Then wrap it back around to 'a'.
        newcord = newcord - ord('z') + ord('a')
      # Update the character in the string.
      s[i] = chr(newcord)
```

```
        # Check if the character is an uppercase letter.
    elif ord('A') <= ord(s[i]) <= ord('Z'):
      # Calculate its new ord number if so.
      newcord = ord(s[i]) + n
      # If the new character has gone beyond 'Z'.
      if newcord > ord('Z'):
        # Then wrap it back around to 'a'.
        newcord = newcord - ord('Z') + ord('A')
      # Update the character in the string.
      s[i] = chr(newcord)
  # Return the updated string.
  return ''.join(s)

# Tests from question.
print(caesar('abcd', 3))
print(caesar('Hello, world!', 2))
```

9. Write a function `sortlist` that takes a list of integers and returns a copy of it sorted. Note that Python has a built-in sort function, but try to solve this problem without using it. For example:

```
> sortlist([3,1,2])
[1,2,3]
> sortlist([10,-9,5,-1,0])
[-9,-1,0,5,10]
```

Solution:

```
# Ian McLoughlin, 2018-03-14
# https://github.com/ianmcloughlin/problems-python-fundamentals

def sortlist(l):
  # Loop through the elements of the list.
  for i in range(len(l)):
    # Loop through the remaining elements of the list.
    for j in range(i+1, len(l)):
      # If the i^th element is greater than the j^th.
      if l[i] > l[j]:
        # Swap them.
        l[i], l[j] = l[j], l[i]
  # Return the sorted list.
  return l

# Tests from question.
```

```
print(sortlist([3,1,2]))
print(sortlist([10,-9,5,-1,0]))
```

10. Write a function `countstr` that takes string and returns, for each character in the string, the number of times the character is contained in it. You might use a dictionary for this purpose. For example:

```
> countstr('aaacbb')
{'a': 3,'c': 1,'b': 2}
> countstr('Hello, world!')
{'H': 1,'e': 1,'l': 3,'o': 2,',': 1,' ': 1,'w': 1,'r': 1,'d': 1,'!': 1}
```

Solution:

```
# Ian McLoughlin, 2018-03-14
# https://github.com/ianmcloughlin/problems-python-fundamentals

def countstr(s):
  # The dictionary that will contain the answer.
  ans = {}
  # Loop through the string.
  for c in s:
    ans[c] = ans.get(c, 0) + 1
  # Return the dictionary.
  return ans

# Tests from question.
print(countstr('aaacbb'))
print(countstr('Hello, world!'))
```

References

[1] Python Software Foundation. Welcome to python.org.