

List numbers one by one

```
20 # Get input from the user
21 number = int(input("Please enter a number 7 to 18 digits: "))
22 # Determine the number of digits in the integer
23 num_digits = len(str(number))
24 # Ensure the number is between 7 to 18 digits
25 if 7 < num_digits and num_digits <= 18:
26     # Use a loop to print each digit
27     for i in range(num_digits):
28         # Print the digit
29         print(num_digits - num_digits, '18 ** num_digits', 10 ** num_digits)
30 # Calculate the power of 10 for the current position
31 power_of_10 = 10 ** (num_digits - 1)
32 # Print the power of 10, power of 10, 'number', number, 'number // power_of_10', number // power_of_10
33 # Extract the leftmost digit
34 digit = number // power_of_10
35 # Print the digit
36 print(digit)
37 # Remove the leftmost digit from the number
38 number %= power_of_10
39 # Decrease the number of digits left to process
40 num_digits -= 1
41 else:
42     print("Number was not 7 to 18 digits.")
```

sentinel loop

```
114 # Initialize total miles and total gallons to zero
115 total_miles = 0
116 total_gallons = 0
117 overall_avg = 0
118 # Start the sentinel-controlled loop, meaning it repeats until it is broken by user input
119 while True:
120     # Get the gallons used from the user, convert to float
121     gallons = float(input("Please enter the gallons used (-1 to stop the loop): "))
122     # Check for the sentinel value and if so escape
123     if gallons == -1:
124         break
125     # Get the miles driven from the user
126     miles = float(input("Please enter the miles driven: "))
127     # Calculate miles per gallon for this tankful
128     mpg = miles / gallons
129     print("The miles/gallon for this tank was {:.1f}".format(mpg))
130 # Update the total miles and total gallons for avg calculation later
131 total_miles += miles
132 total_gallons += gallons
133 # Calculate the overall average miles per gallon
134 if total_gallons != 0:
135     overall_avg = total_miles / total_gallons
136 print("The overall average miles/gallon was {:.1f}".format(overall_avg))
```

palindrome

```
149 # 3.12 (Palindromes) A palindrome is a number, word or text phrase that reads the same
150 # Get input from the user
151 number = int(input("Please enter a five-digit integer: "))
152 # Ensure the number is five digits
153 if 10000 <= number and number <= 99999:
154     # Extract the digits
155     digit1 = number // 10000
156     digit2 = (number // 1000) // 1000
157     digit3 = (number // 100) // 10000, 'divide by 1000', (number // 10000) // 1000
158     digit4 = (number // 100) // 100
159     digit5 = (number // 10) // 10
160     print("digit1", number // 10000)
161     print("digit2", number // 1000, 'divide by 1000', (number // 10000) // 1000)
162     print("digit3", number // 1000, 'divide by 100', (number // 10000) // 100)
163     print("digit4", number // 100, 'divide by 10', (number // 10000) // 10)
164     print("digit5", number // 10, 'divide by 10', (number // 10000) // 10)
165     # Check if the number is a palindrome
166     if digit1 == digit5 and digit2 == digit4:
167         print("{} is a palindrome.".format(number))
168     else:
169         print("{} is not a palindrome.".format(number))
170 else:
171     print("Please enter a five-digit integer.")
```

consecutives in list

```
190 i = 1
191 total = 0
192 consecutive_3_14 = 0
193 consecutive_3_141 = 0
194 iteration_for_3_14 = 0
195 iteration_for_3_141 = 0
196
197 for denominator in range(1, 6000, 2):
198     if i % 2 == 0:
199         total -= 4/denominator
200     else:
201         total += 4/denominator
202
203 # Check for consecutive 3.14 approximations
204 if str(total)[4] == '3.14':
205     consecutive_3_14 += 1
206     if consecutive_3_14 == 2 and not iteration_for_3_14:
207         iteration_for_3_14 = i
208         print('hit 3.14', i)
209     else:
210         consecutive_3_14 = 0
211
212 # Check for consecutive 3.141 approximations
213 if str(total)[5] == '3.141':
214     consecutive_3_141 += 1
215     if consecutive_3_141 == 2 and not iteration_for_3_141:
216         iteration_for_3_141 = i
217         print('hit 3.141', i)
218     else:
219         consecutive_3_141 = 0
220
221 # Print the results
222 i += 1
223 print('i:', i, 'total', total)
```

splicing

```
18 print("subset1:", subset1_grades)
19 # Just the last element
20 subset2 = subset1_grades[-1:]
21 print("subset2:", subset2)
22 # reversing our list with splicing
23 grades_rev = grades[::-1]
24 print("grades_rev:", grades_rev)
25 # return everything except for the last two elements
26 subset3 = grades[:-2]
27 print("subset3:", subset3)
28 # return every other number
29 every_other_grade = grades[::2]
```

moving avg

```
def moving_avg(data, window_size):
    # Initialize a list to hold the moving averages
    moving_averages = []
    # Iterate over the data
    for i in range(len(data)):
        # Calculate the moving average for the current point
        start_index = max(0, i - window_size + 1)
        end_index = i + 1
        moving_averages.append(sum(data[start_index:end_index]) / (end_index - start_index))
    return moving_averages
```

arithmetic

```
def arithmetic_sequence(a, d, n):
    # Initialize a list to hold the arithmetic sequence
    sequence = []
    # Iterate over the range of n
    for i in range(n):
        # Calculate the nth term of the arithmetic sequence
        term = a + d * i
        sequence.append(term)
    return sequence
```

pi

```
def calculate_pi(n_digits):
    # Initialize a list to hold the digits of pi
    digits = []
    # Iterate over the range of n_digits
    for i in range(n_digits):
        # Calculate the next digit of pi
        # (This is a simplified version of the actual algorithm)
        digit = (i * 10) % 10
        digits.append(digit)
    return digits
```

odd or even

```
def is_odd_or_even(n):
    # Check if the number is odd or even
    if n % 2 == 0:
        return "Even"
    else:
        return "Odd"
```

multiples

```
def find_multiples(n, limit):
    # Find multiples of n up to limit
    multiples = []
    for i in range(1, limit // n + 1):
        multiples.append(n * i)
    return multiples
```

formatting

```
def format_data(data):
    # Format the data for display
    formatted_data = []
    for item in data:
        formatted_data.append(f"{item:.2f}")
    return formatted_data
```