3. Loops

October 17, 2022

1 Introduction

- We can iterate over a sequence of elements using loops.

2 While Loop

- Executes a block of statements repeatedly as long as the condition is True.

Syntax:

```
while boolean_expression :
    statement 1
    statement 2
    statement 3
    .
    .
    statement n
```

[ ]:
```
c = 0

while c < 10:
    print(c)
    c = c + 1
```

2.1 Example

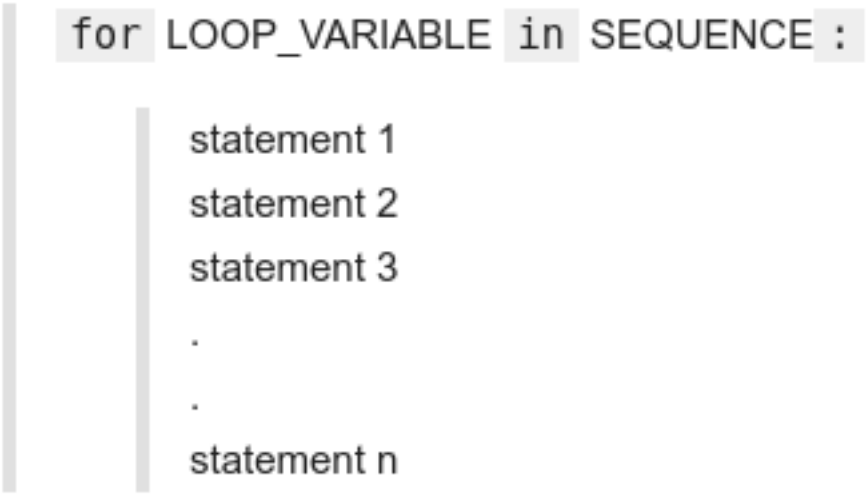
Write a program Find out whether the given number is Armstrong number or not. Armstrong number is a number that is equal to the sum of the cubes of its own digits.

[ ]:
```python
num = int(input('Enter a number: '))
value = 0

# find the sum of the cube of each digit
temp = num
while temp > 0:
    digit = temp % 10
    value = value + digit ** 3
    temp = temp // 10

# display the result
if num == value:
    print('its armstrong no')
else:
    print('not armstrong no')
```

3 For Loop

- Is an iterator based loop, which steps through the items of iterable objects like lists, tuples, string and executes a piece of code repeatedly for a number of times, based on the number of items in that iterable object.

Syntax:



3.1 For loop on sequence of numbers

The `range()` function: - We can generate a sequence of numbers using `range()` function.

- For example `range(10)` will generate numbers from 0 to 9 (10 numbers).
- We can also define the `start`, `stop` and `step size` as `range(start, stop, step_size)`. `step_size` defaults to 1 if not provided.

- The range object is "lazy" in a sense because it doesn't generate every number that it "contains" when we create it.
- This function does not store all the values in memory; it would be inefficient. So it remembers the start, stop, step size and generates the next number on the go.
- To force this function to output all the items, we can use the function `list()`.

[ ]:
```
range(10) # range(stop)
```

[ ]:
```
range(10,20) #range(start, stop)
```

[ ]:
```
range(10,20,2) # range(start, stop, step_size)
```

[ ]:
```
list(range(10,20)) #range(start, stop)
```

[ ]:
```
list(range(10,20,2))
```

[ ]:
```
list(range(10))
```

[ ]:
```
# print nos 0 to 9 using for loop
for i in range(10):
    print(i)
    print('-')
```

[ ]:
```
for i in range(10,20):
    print(i)
```

[ ]:
```
for i in range(10,20,2):
    print(i)
```

4 Break and Continue

- Either skip a single iteration (as many times you want) using `continue`, or you can break out of the loop entirely using `break`.

4.1 `break`

- If you want to loop until something occurs, but you're not sure when that might happen, you can use an infinite loop with a `break` statement.

[ ]:
```
# Capitalize a string
while True:
    word = input("String to capitalize [type q to quit]: ")
    if word == "q":
        break
    print(word.capitalize())
```

4.2 `continue`

- Without break out of a loop, skip ahead to the next iteration.

[ ]:
```
# Square only odd numbers
while True:
    value = input("Integer [press q to quit] : ")

    if value == 'q':       # quit
        break

    number = int(value)

    if number % 2 == 0:    # an even number
        continue
    print(number, "squared is", number*number)
```

5 A special `else` clause – after while and for loops

- If while loop ended normally (no break call), control passes to an optional else.

[ ]:
```
i = 1
while i <  4:
    print(i)
    i += 1
    if i == 5:
        break
else:
    print("i is no longer less than 6")
```

[ ]:
```
for i in range(1,10,2):
     if i % 2 == 0:
        print('Found even number', i)
        break
else:
    print('No even number found')
```

6 Example

- Write a program to find factors of a number.

[ ]:
```
# take input from the user
x = int(input("Enter a number: "))

print("The factors of",x,"are:")

for i in range(1, x + 1):
```

```
    if x % i == 0:
        print(i)
```