Python - Upto Loops

January 10, 2023

1 Basics of Python Programming Language

[7]:
```
print("Pavan")
```

```
Pavan
```

[8]:
```
print(10)
```

```
10
```

[9]:
```
print(name)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [9], in <cell line: 1>()
----> 1 print(name)

NameError: name 'name' is not defined
```

[10]:
```
print(pavan)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [10], in <cell line: 1>()
----> 1 print(pavan)

NameError: name 'pavan' is not defined
```

[11]:
```
print('pavan')
```

```
pavan
```

[12]:
```
print(12.2)
```

```
12.2
```

1.1 Data types in Python

- int
- float
- str
- bool
- list
- tuple
- set
- dict

1.1.1 int

- Used to represent integers
- integers - positive, negative, 0

[14]:
```
a = 10
print(a)
print(type(a))
```

```
10
<class 'int'>
```

[15]:
```
a = -10
print(a)
print(type(a))
```

```
-10
<class 'int'>
```

[16]:
```
a = 0
print(a)
print(type(a))
```

```
0
<class 'int'>
```

1.1.2 float

- any number with point value
- +ve, -ve, 0

[17]:
```
f = 10.2
print(f)
print(type(f))
```

```
10.2
<class 'float'>
```

[18]:
```
f = -10.2
print(f)
```

```
print(type(f))
```

```
-10.2
<class 'float'>
```

[20]:
```
f = 0.0
print(f)
print(type(f))
```

```
0.0
<class 'float'>
```

1.1.3 str

- everything put within single quotes or double quotes or triple quotes will be treated as a string.

[22]:
```
name = "Pavan"
print(name)
print(type(name))
```

```
Pavan
<class 'str'>
```

[23]:
```
name = 'Pavan'
print(name)
print(type(name))
```

```
Pavan
<class 'str'>
```

[24]:
```
name = '''Pavan'''
print(name)
print(type(name))
```

```
Pavan
<class 'str'>
```

[27]:
```
a = 10
b = 20
print(a+b)
```

```
30
```

[28]:
```
a = '10'
b = '20'
print(a+b)
```

```
1020
```

1.1.4 bool

- Boolean
- True, False

[29]:
```
10 > 20
```

[29]:
```
False
```

[30]:
```
2 > 1
```

[30]:
```
True
```

[31]:
```
7 <= 7
```

[31]:
```
True
```

1.1.5 List

- List is an ordered collection of elements or items
- List items can be value that belongs python data type
- List elements are enclosed using square braces
- There are two types of lists in Python
- Homogeneous list, Heterogeneous list
- Homogeneous list -> A list that contains elements of same type
  - Ex: [10, 20, 30], [10.2, 20.2, 30.2], ['this', 'is', 'python']
- Heterogeneous list -> A list that contains elements of different types
  - Ex: [10, 2.2, 'hello world', True]
- List elements can be accessed using indexes
- List index starts with 0

[1]:
```
lst = [10,20,30]
print(lst)
print(type(lst))
```

```
[10, 20, 30]
<class 'list'>
```

[2]:
```
pavan = [10,20,30]
print(pavan)
print(type(pavan))
```

```
[10, 20, 30]
<class 'list'>
```

[3]:
```
a =   [10, 20, 30]
#index 0   1   2
print(a)
```

```
[10, 20, 30]
```

[4]:
```
# Accessing list elements
lst = [10, 20, 30]
print(lst[1])
```

```
20
```

[5]:
```
x = [12, 19, 71, -14, 6, 8, 13, -76]
#in  0   1   2    3   4  5  6    7
print(x[2] + x[3] - x[4] + x[6]) #64
#      71 - 14 - 6 + 13
```

```
64
```

[6]:
```
my_list = [10, 12.2, 'hello', 67.2, '123', True, 795]
print(my_list[2])
print(type(my_list[2]))
print(my_list[4])
print(type(my_list[4]))
print(my_list[5])
print(type(my_list[5]))
```

```
hello
<class 'str'>
123
<class 'str'>
True
<class 'bool'>
```

1.1.6 tuple()

- Is also an ordered collection of elements
- Elements are enclosed within round braces (parentheses)
- tuple elements can also be accessed using indexes.

[7]:
```
x = (10, 20, 30)
#in  0   1   2
print(x)
print(type(x))
```

```
(10, 20, 30)
<class 'tuple'>
```

[8]:
```
x = (10, 20, 30)
#in  0   1   2
print(x[0])
```

```
10
```

1.1.7 set

- set is an unordered collection of unique elements

- set elements are enclosed using curly/flower braces
- set will not hold duplicates
- set elements cannot be accessed using indexes

[9]:
```
l = [10, 20, 30, 40, 10]
print(l)
print(type(l))
```

```
[10, 20, 30, 40, 10]
<class 'list'>
```

[11]:
```
s = {10, 20, 30, 40, 10, 10, 10, 10}
print(s)
print(type(s))
```

```
{40, 10, 20, 30}
<class 'set'>
```

[12]:
```
fruits = {'apple', 'orange', 'apple', 'mango', 'kiwi', 'kiwi', 'mango'}
print(fruits)
```

```
{'apple', 'orange', 'kiwi', 'mango'}
```

[13]:
```
s1 = {10, 20, 30}
s2 = {30, 40, 50, 10}
print(s1.intersection(s2))
```

```
{10, 30}
```

[14]:
```
my_set = {'a', 'a', 'a', 'a', 'a', 'a', 'a'}
print(my_set)
```

```
{'a'}
```

1.1.8 dictionary

- dictionaries are used to store elements that are in the forms of pairs
- a dictionary element should contain a key and a value
- Ex:
- word: definition
- actor: no.of films
- actor: best picture
- batsmen: no. of runs
- bowler: no. of wickets
- author: best seller
- dictionary elements are enclosed using curly braces
- Using key we can get the value

[3]:
```
# Let's say that we want to keep the data of batsment:centuries scored
# we can use the dictionary for this type of data where
# key --> Batsmen
```

```
# value --> Centuries scored
d = {'sachin': 100,
     'kohli': 72,
    'ponting': 71,
    'sangakkara': 63,
    'kallis': 62,
     'hashim amla': 55}
print(d)
print(type(d))
```

```
{'sachin': 100, 'kohli': 72, 'ponting': 71, 'sangakkara': 63, 'kallis': 62,
'hashim amla': 55}
<class 'dict'>
```

[4]:
```
# accessing dictionary elements
# we can use dictionary_name[key] to get the value
d = {'sachin': 100,
     'kohli': 72,
    'ponting': 71,
    'sangakkara': 63,
    'kallis': 62,
     'hashim amla': 55}
print(d['sachin'])
print(d['hashim amla'])
```

```
100
55
```

[5]:
```
# We can add a new record to the existing dictionary as follows
d['jayawardene'] = 54
print(d)
```

```
{'sachin': 100, 'kohli': 72, 'ponting': 71, 'sangakkara': 63, 'kallis': 62,
'hashim amla': 55, 'jayawardene': 54}
```

1.2 Variables

- Variables are container of data
- Variable holds a value

1.2.1 Rules to create variable names in Python

- Varialbe name can be alphanumeric, but it should not start with a digit.
- No other special character is allowed with variable name except underscore (_).
- Variable name can start with an underscore and underscore itself can be used as a variable name.
- Variable name should not contain any whitespaces within.
- Keywords should not be used as variable names.
- Variable names are case sensitive, means NUM, num, Num will be treated as different variables due to the differences in case even though they are spelled the same.

Varialbe name can be alphanumeric, but it should not start with a digit.

[1]:
```
person1 = 'Berlin'
print(person1)
```

```
Berlin
```

[2]:
```
1stperson = 'Berlin'
print(1stperson)
```

```
  Input In [2]
    1stperson = 'Berlin'
    ^
SyntaxError: invalid decimal literal
```

No other special character is allowed within variable name except underscore (_).

[3]:
```
person_1 = 'Berlin'
print(person_1)
```

```
Berlin
```

[4]:
```
person@1 = 'Berlin'
print(person@1)
```

```
  Input In [4]
    person@1 = 'Berlin'
    ^
SyntaxError: cannot assign to expression here. Maybe you meant '==' instead of␣
 ↪'='?
```

Variable name can start with an underscore and underscore itself can be used as a variable name.

[5]:
```
_name = 'Pavan'
print(_name)
```

```
Pavan
```

[6]:
```
_ = 10
print(_ + _ + _)
```

```
30
```

Variable name should not contain any whitespaces within.

[7]:
```
first name = 'John'
last name = 'Smith'
```

```
print(first name)
```

```
  Input In [7]
    first name = 'John'
          ^
SyntaxError: invalid syntax
```

[8]:
```
first_name = 'John'
last_name = 'Smith'
print(first_name)
print(last_name)
```

```
John
Smith
```

Keywords should not be used as variable names.

[9]:
```
in = 10
print(in)
```

```
  Input In [9]
    in = 10
    ^
SyntaxError: invalid syntax
```

[10]:
```
in1 = 10
print(in1)
```

```
10
```

Variable names are case sensitive

[12]:
```
num = 10
NUM = 20
Num = 30
print(Num)
```

```
30
```

1.2.2 Naming Conventions

[15]:
```
num1 = 10
num2 = 20
product = num1 * num2
print(product)
```

```
200
```

1.3 Operators

Terminology of an Operation:
Every operation consists two parts
1. Operand(s) 2. Operator

Operand(s): On which the operation is being performed
Operator: The one which is performing the operation

Ex:
a + b
Here Operands: a, b
Operator: +

Types of Operators: - Arithmetic Operators - Relational or Comparison Operators - Logical Operators - Assignment Operators - Bitwise Operators - Membership Operators - Identity Operators

1.3.1 Arithmetic Operators

- + –> Addition –> Sum
- - –> Subtraction –> Difference
- * –> Multiplication –> Product
- / –> Division –> Quotient (Actual)
- // –> Floor or Integer Division –> Quotient (Integer)
- % –> Modulo Division –> Remainder
- ** –> Exponentiation –> Power

Addition (+)

- Used on two integers
- Used on two floating values
- Used on two sequence types (list, str, tuple)
- When used on sequence types + symbol acts as concatenation operator

on two integers

[16]:
```
10 + 20
```

[16]:
```
30
```

[17]:
```
a = 10
b = 20
print(a + b)
```

```
30
```

On two sequence types

[18]:
```
print('hello'+'world')
```

```
helloworld
```

[19]:
```
ls1 = [10, 20, 30]
ls2 = [40, 50, 60]
print(ls1 + ls2)
```

```
[10, 20, 30, 40, 50, 60]
```

Subtraction (-)

- On two integers
- On two floating point values

[20]:
```
a = 10
b = 20
print(b - a)
```

```
10
```

Multiplication (*)

- On integers
- On two floting point values
- On one interger and one sequence type (list, tuple, string)
- When used on a number and a sequence type * symbol acts as a repetition operator

[21]:
```
a = 10
b = 2
print(a * b)
```

```
20
```

[26]:
```
n = 5
name = 'pavan\n'
print(n * name)
```

```
pavan
pavan
pavan
pavan
pavan
```

[24]:
```
lst = [0]
print(20 * lst)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

[25]:
```
lst = [1, 2, 3]
print(lst * 5)
```

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

[28]:
```
print('hel\nlo\nwo\nrld')
```

```
hel
lo
wo
rld
```

Division (/)

- Used to produce actual quotient of a division operation.

[ ]:
```
2)12(6 --> quotient
  12
-----
  0 --> Remainder
----
```

[29]:
```
print(10/2)
```

```
5.0
```

[30]:
```
print(10/4)
```

```
2.5
```

[31]:
```
print(100/10)
```

```
10.0
```

integer division (floor division) (//)

- Produces floored quotient
- Flooring - Rounding down to nearest integer

[32]:
```
print(10/4)
```

```
2.5
```

[33]:
```
print(10//4)
```

```
2
```

[34]:
```
print(11/5)
```

```
2.2
```

[35]:
```
print(11//5)
```

```
2
```

[ ]:
```
2.9 --> 2 --> Rounding down
14.7 --> 14
```

[36]:
```
-10/4
```

[36]: -2.5

[37]:
```
-10//4
```

[37]: -3

Modulo division (%)

- Remainder

[38]:
```
print(10%3)
```

1

[39]:
```
print(14%9)
```

5

[40]:
```
print(100%51)
```

49

[41]:
```
print(100%49)
```

2

[48]:
```
a = 789
b = 456
print((a%10) + (b%10))
```

15

[47]:
```
a%10
```

[47]: 9

exponentiation operator (**)

- power
- a ** b –> a power b

[49]:
```
print(2**3)
```

8

[50]:
```
print(10**5)
```

100000

[51]:
```
#Area of a square
side = 4
area = side * side
print(area)
```

16

[52]:
```
a = 2
b = 3
res = a**2 + b**2 + 2*a*b
print(res)
```

25

[53]:
```
a = 2
b = 3
res = (a+b)**2
print(res)
```

25

1.3.2 Evaluation of Arithmetic Expressions (Operator Precedence and Associativity)

We know that in order to evaluate an arithmetic expression we have to follow the BODMAS principle.

BODMAS B –> Brackets
O –> Order
D –> Division
M –> Multiplication
A –> Addition
S –> Subtraction

In Python we call it as PEMDAS
P –> Parentheses
E –> Exponentiation
M –> Multiplication
D –> Division
A –> Addition
S –> Subtraction

Order of Precedence in Arithmetic Operators

- First Priority –> **
- Second Priority –> *, /, //, %
- Third Priority –> + , -

[1]:
```
print(2 + 2 // 2)
```

3

[2]:
```
print(10 + 2 ** 2 * 3 - 6 // 2)
# 19
```

19

[ ]:
```
print(4*3//2)
```

[3]:
```
print(3 + 2 * 3 - 4 // 2 + 2 ** 2 + 7)
# 18
```

18

[4]:
```
print(10 * 2 + 2 * 4 // 2 - 1 * 4 % 2 ** 2)
# 24
```

24

[5]:
```
print(33 - 10 * 2 + 14 // 7 - 3 + 2 % 1 + 6)
# 18
```

18

[6]:
```
print(2 ** 2 + 3 * 2 ** 2 % 2 - 3 + 8 ** 2)
# 65
```

65

Associativity

- If an expression contains operators that are having same precedence, then associativity must be applied
- Types of Associativity
  - Left to right
  - Right to left
- All arithmetic Opeartors in Python follows Left to Right Associativity except exponentiation (**)
- Exponentiation follows Right to Left Associativity
- Binding (Left binding - L - R)
- Right (R - L)

[7]:
```
print(3 * 4 % 2 // 2 * 3)
# 0
```

0

[8]:
```
print(2 + 3 - 6 + 11 - 14 + 22)
# 18
```

18

[10]:
```
print(2 ** 3 ** 2)
```

```
512
```

1.3.3 Relational or Comparison Operators

- Used in decision making
- >
- >=
- <
- <=
- ==
- !=
- The result is always a boolean value (True or False)

[11]:
```
print(10 > 5)
```

```
True
```

[12]:
```
print(10 > 20)
```

```
False
```

[13]:
```
print(10 >= 10)
```

```
True
```

[14]:
```
print(5 <= 10)
```

```
True
```

[15]:
```
print(11 <= 11)
```

```
True
```

[16]:
```
print(10 >= 20)
```

```
False
```

[17]:
```
print(10 == 10)
```

```
True
```

[18]:
```
print(1 != 2)
```

```
True
```

[19]:
```
print(10 != 20)
```

```
True
```

[20]:
```
print(1 != 1)
```

```
False
```

Continuous relational expressions

[21]:
```
print(10 < 20 < 30)
```

```
True
```

[23]:
```
a = 15
b = 10
c = 20
# print(a<b<c)
print(b<a<c) #10<15<20
```

```
True
```

1.3.4 Logical Operators

- Used to combine two or more relational expressions
- The result is always a boolean value
- Logical AND: and
- Logical OR: or
- Logical NOT: not

Logical AND (and)

- True if both left and right operands are True, False otherwise

[24]:
```
print(10<20 and 20<30)
```

```
True
```

[25]:
```
print(10<20 and 40<30)
```

```
False
```

[28]:
```
print(10>2 and 20>10 and 2==2 and 4!=5 and 5>=4)
```

```
True
```

[29]:
```
print(10>2 and 20>10 and 2==3 and 4!=5 and 5>=4)
```

```
False
```

Logical OR (or)

- True, even if one of the operands is True, False otherwise

[30]:
```
print(10<20 or 20<30)
```

```
True
```

[34]:
```
print(10<20 or 20>30)
#print(True or False)
```

```
True
```

[32]:
```
print(10>20 or 20>30)
# print(False or False)
```

```
False
```

[33]:
```
10>20
```

[33]:
```
False
```

Logical NOT (not)

- Unary Operator
- Inverse Truth
- When the expression is True, it will give you False
- When the expression is False, it will give you True

[36]:
```
not (10>5)
# not True
```

[36]:
```
False
```

[39]:
```
not (10 > 11)
# not False
```

[39]:
```
True
```

1.3.5 Assignment Operators

1. Used to assign values to variables
2. Assignment Operators are
   - = –> Assign
   - += –> Add and assign
   - -= –> Subtract and assign
   - *= –> Multiply and assign
   - /= –> Divide and assign (the actual quotient)
   - //= –> Floor divide and assign (the floored quotient)
   - %= –> Modulo divide and assign (remainder)
   - **= –> **power and assign (a**b)
3. += –> Add and assign –> Adds right operand to left operand and reassigns the result to left operand
4. a += b –> Reassigns a with a+b

5. y *= x –> Reassigns y with y*x

Examples

[5]:
```
a = 10
a += 5
print(a)
```

```
15
```

[6]:
```
a = 100
a -= 50
print(a)
```

```
50
```

[7]:
```
a = 10
b = 5
a *= b
print(a)
print(b)
```

```
50
5
```

[8]:
```
a = 10
b = 5
b *= a
print(a)
print(b)
```

```
10
50
```

[9]:
```
x = 100
x /= 25
print(x)
```

```
4.0
```

[11]:
```
y = 51
y //= 16
print(y)
```

```
3
```

[12]:
```
g = 2
g **= 3
print(g)
```

```
8
```

[13]:
```
z = 100
z %= 20
print(z)
```

```
0
```

[4]:
```
a = 10 # the value of a 10
print(a)
a = 20 # # the value of a 20
a = 30 # the value of a 30
print(a * a)
```

```
10
900
```

[15]:
```
a = 3
b = 4
c = 5
a += b # a b c = 7 4 5
b += a # a b c = 7 11 5
c += b # a b c = 7 11 16
c += a # a b c = 7 11 23
a *= b # a b c = 77 11 23
c -= b # a b c = 77 11 12
print(b, c, a) # 77 11 12 #
```

```
11 12 77
```

[17]:
```
x = 3
y = 2
z = 4
x *= y  # x y z = 6 2 4
y *= z  # x y z = 6 8 4
z += x  # x y z = 6 8 10
x += z  # x y z = 16 8 10
x //= y # x y z = 2 8 10
z += x  # x y z = 2 8 12
print(x, y, z) # 5 8 12 # 2 8 12 #
```

```
2 8 12
```

Single line assignments

[18]:
```
a = 10
b = 20
c = 30
print(a, b, c)
```

```
10 20 30
```

[19]:
```
a, b, c = 10, 20, 30
print(a)
print(b)
print(c)
```

```
10
20
30
```

[20]:
```
a = 10
b = 10
c = 10
print(a, b, c)
```

```
10 10 10
```

[21]:
```
a = b = c = 10
print(a, b, c)
```

```
10 10 10
```

1.3.6 swapping of two values in Python

Using third variable

[ ]:
```
a = 10
b = 20
print(f'Before swapping\na: {a}\nb: {b}')
c = a
a = b
b = c
print(f'\nAfter swapping\na: {a}\nb: {b}')
```

Without using third variable

[5]:
```
a = 10
b = 20
print(f'Before swapping\na: {a}\nb: {b}')
a, b = b, a
print(f'\nAfter swapping\na: {a}\nb: {b}')
```

```
Before swapping
a: 10
b: 20

After swapping
a: 20
b: 10
```

1.3.7 Decimal Number System (Base 10)

- 100, 5465456946566 –> Decimal Number System

- Deci –> Ten (10)
- Digits used in the system –> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 –> 10

1.3.8 Binary Number System (Base 2)

- There are only two digits –> 0, 1
- 2 (and it's powers) play a significant role in binary

Decimal to Binary Conversion Decimal Number: 23
Binary Represe: 10111

- Find out the nearest power of 2 which is <= given number
- Put a 1 below the numbers that you can use to add upto the given number
- Put a 0 below the numbers that you can't use

16 8 4 2 1
1 0 1 1 1 –> This the binary representation of 23.
(10111)

Decimal: 49
Binary: 110001

32 16 8 4 2 1
1 1 0 0 0 1 –> This the binary of 49.
(110001)

Decimal: 75
Binary: 1001011

64 32 16 8 4 2 1
1 0 0 1 0 1 1 –> This is the binary of 75
(1001011)

Halving
Decimal: 23
Binary: ?
23 –> 1
11 –> 1
5 –> 1
2 –> 0
1 –> 1
- The number is even write a 0 - The number is odd write a 1

Write the results bottom up
(10111)

Decimal: 75
Binary: 1001011

75 –> 1
37 –> 1
18 –> 0
9 –> 1

4 –> 0
2 –> 0
1 –> 1

Bottom up (1001011)

Binary to Decimal Conversion ***Example 1:***

Binary: 10111
Decimal: 23

To convert any binary number into decimal form do the following

- Seperate the bits as follows and
- Write the powers of 2 (starting from 2^0) from right to left under each bit

1	0	1	1	1
16	8	4	2	1

And add all the numbers under 1 to get the decimal representation

16 + 4 + 2 + 1 = 23

Some more examples

Example 2:

Binary: 1101101
Decimal: 109

Seperating bits

1	1	0	1	1	0	1
64	32	16	8	4	2	1

Adding all the numbers that are under 1 to get the decimal representation

64 + 32 + 8 + 4 + 1 = 109

Example 3:

Binary: 1111011
Decimal:

Seperating bits

1	1	1	1	0	1	1
64	32	16	8	4	2	1

Adding all the numbers that are under 1 to get the decimal representation

$64 + 32 + 16 + 8 + 2 + 1 = 123$

1.3.9 Bitwise Operators

- Operates on bit level
- Requires knowledge on Binary Number System.
- & –> Bitwise AND
- –> Bitwise OR
- ^ –> Bitwise XOR
- « –> Left shift
- » –> Right shift

Bitwise AND (&)

- Compares each bit of first operand to the corresponding bit of second operand and sets the result bit to 1, if and only if both bits are 1, otherwise to 0

[42]:
```
a = 11
b = 12

# 11 -->  1011
# 12 --> 1100

# 1011
# 1100
# ----
# 1000 --> 8
print(a & b)
```

```
8
```

Bitwise OR (|)

- Compares each bit of first operand to the corresponding bit of second operand and sets the result bit to 1, even if one of the bits is 1, otherwise to 0

[43]:
```
a = 11
b = 12

# 1011
# 1100
# ----
# 1111 --> 15
print(a | b)
```

```
15
```

Bitwise XOR

- Compares each bit of first operand to the corresponding bit of
  second operand and sets the result bit to 1, if one bit is 0 and the other is 1 or vice versa (alternate bits) otherwise to 0.

[44]:
```
a = 11
b = 12
# 1011
# 1100
# ----
# 0111 --> 7
print(a ^ b)
```

```
7
```

A = 9 B = 14

A	B	A&B	A|B	A^B
1	1	1	1	0
0	1	0	1	1
0	1	0	1	1
1	0	0	1	1
		8	15	7

Bit Shortage:

- In case of a bit shortage, we are allowed to add bits (0s) on the LEFT SIDE (Preceding bits) of the operand.

Left shift («)

- Left shift <–> multiplying
- a « b <–> a * 2 ** b

[64]:
```
print(3 << 2)
# 12
```

```
12
```

[65]:
```
print(7 << 1)
# 7 * 2 = 14
```

```
14
```

[66]:
```
print(11 << 4)
# 11 * 16
```

```
176
```

Right shift (»)

- Right shift <–> dividing
- a » b <–> a // 2 ** b

[67]:
```
print(11 >> 2)
# 11 // 2 ** 2 --> 11 // 4 --> 2
```

```
2
```

[68]:
```
print(25 >> 1)
# 25 // 2 ** 1 --> 25 // 2 --> 12
```

```
12
```

[69]:
```
print(76 >> 4)
# 76 // 2 ** 4 --> 76 // 16 --> 4
```

```
4
```

1.3.10 Membership Operators

- in
- not in
- Tells if the given element is a member of an iterable
- Iterables: Everything that can be looped over is an iterable in python
- Examples of iterable objects: list, string, tuple, set, range
- Result of a membership operator is always a boolean value (True or False)

[70]:
```
print('h' in 'hello')
```

```
True
```

[71]:
```
print('H' in 'hello')
```

```
False
```

[72]:
```
print('hell' in 'hello')
```

```
True
```

[73]:
```
print('helo' in 'hello')
```

```
False
```

[74]:
```
print('llo' in 'hello')
```

```
True
```

[75]:
```
print(' ' in 'hello world')
```

```
True
```

[76]:
```
my_list = [10, 20, 30]
print(10 in my_list)
```

```
True
```

[77]:
```
my_list = [10, 20, 30]
print(100 in my_list)
```

```
False
```

[78]:
```
print(1 in 100)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [78], in <cell line: 1>()
----> 1 print(1 in 100)

TypeError: argument of type 'int' is not iterable
```

[79]:
```
print('1' in '100')
```

```
True
```

[80]:
```
print(2 in 12.2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [80], in <cell line: 1>()
----> 1 print(2 in 12.2)

TypeError: argument of type 'float' is not iterable
```

[81]:
```
print('h' not in 'hello')
```

```
False
```

[82]:
```
print(100 not in [10, 20, 30])
```

```
True
```

[85]:
```
# Vowel or Consonant
ch = input("Enter a character")
if ch == 'a' or ch == 'e' or ch == 'i' or ch == 'o' or ch == 'u':
    print('Vowel')
else:
    print('Consonant')
```

```
Enter a characterA
Consonant
```

[89]:
```
ch = input("Enter a character: ") # Z
vowels = ['a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U']
if ch in vowels: # if False
    print('Vowel')
else:
    print('Consonant')
```

```
Enter a character: Z
Consonant
```

[92]:
```
# Using Membership Operator
ch = input("Enter a character: ")
vowels = 'aeiouAEIOU'
if ch in vowels:
    print('Vowel')
else:
    print('Consonant')
```

```
Enter a character: O
Vowel
```

[99]:
```
# Program to find if the given character is
# - digit (0 - 9)
# - alphabet (a - z or A - Z)
# - special character (&*%^$...)
ch = input('Enter a character: ')
digits = '0123456789'
alpha = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
if ch in digits:
    print('Digit')
elif ch in alpha:
    print('Alphabet')
else:
    print('Special Character')
```

```
Enter a character: A
Alphabet
```

1.4 Input Reading

- To read integers - int(input())
- To read point values - float(input())
- To read strings - input()

[63]:
```
a = int(input("Enter a number"))
b = int(input("Enter another number"))
c = a + b
print(c)
```

```
Enter a number10
Enter another number20
30
```

[65]:
```
# area of a square
side = int(input("Enter side length: ")) #input
area = side * side #process
print(area) # output
```

```
Enter side length: 15
225
```

[61]:
```
# area of a rectangle
l = int(input())
b = int(input())
area = l*b
print(area)
```

```
3
4
12
```

1.4.1 Single line input reading and Multiline input reading

[19]:
```
a = int(input())
b = int(input())
c = int(input())
print(a + b + c)
```

```
10 20 30
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [19], in <cell line: 1>()
----> 1 a = int(input())
      2 b = int(input())
      3 c = int(input())
```

```
ValueError: invalid literal for int() with base 10: '10 20 30'
```

[22]:
```
# reading two or more integers in a single line
a, b, c = map(int, input().split())
print(a + b + c)
```

```
10 20 30
60
```

[24]:
```
# reading two or more integers in a single line
a, b, c, d = map(int, input().split())
print(a + b + c + d)
```

```
10 20 30 40
100
```

[25]:
```
# reading two or more integers in a single line
a, b = map(int, input().split())
print(a + b)
```

```
10 20
30
```

[ ]:
```
# reading two or more floating values in a single line
a, b = map(float, input().split())
print(a + b)
```

[26]:
```
# reading two or more strings values in a single line
name1, name2, name3 = map(str, input().split())
print(f"Name 1: {name1}")
print(f"Name 2: {name2}")
print(f"Name 3: {name3}")
```

```
vikram rolex tina
Name 1: vikram
Name 2: rolex
Name 3: tina
```

1.5 Output formatting

1.5.1 Normal Approach

[67]:
```
a = int(input())
b = int(input())
c = a + b
print("Sum is:", c)
```

```
10
20
Sum is: 30
```

[69]:
```
a = int(input())
b = int(input())
c = a + b
print("Sum of a and b is:", c)
```

```
10
20
Sum of a and b is: 30
```

[73]:
```
a = int(input()) #10
b = int(input()) #20
c = a + b
print("Sum of", a, "and", b, "is: ", c)
```

```
10
20
Sum of 10 and 20 is:  30
```

1.5.2 Using .format method

[74]:
```
a = int(input()) #10
b = int(input()) #20
c = a + b
print("Sum of {} and {} is {}".format(a, b, c))
```

```
10
20
Sum of 10 and 20 is 30
```

[1]:
```
# area and perimeter of a square
side = int(input())
area = side * side
peri = 4 * side
print("Area of a square with side length {} is {}".format(side, area))
print("Perimeter of a square with side length {} is {}".format(side, peri))
```

```
8
Area of a square with side length 8 is 64
Perimeter of a square with side length 8 is 32
```

1.5.3 Using f strings (Works on Python version 3.6 or higher)

[75]:
```
a = int(input()) #10
b = int(input()) #20
c = a + b
print(f"Sum of {a} and {b} is {c}")
```

```
10
20
Sum of 10 and 20 is 30
```

[2]:
```
# area and perimeter of a square
side = int(input())
area = side * side
peri = 4 * side
print(f"Area of a square with side length {side} is {area}")
print(f"Perimeter of a square with side length {side} is {peri}")
```

```
8
Area of a square with side length 8 is 64
Perimeter of a square with side length 8 is 32
```

1.5.4 Adjusting a float pointe value to certian (n) digits after point

[3]:
```
pi = 3.141592653
# Using % formatting
print("%.2f"%pi)
# here we are using %f to represent a floating point value and
# .2 to adjust it to 2 decimal places after point
```

```
3.14
```

[4]:
```
pi = 3.141592653
# Using .format()
print("{:.2f}".format(pi))
```

```
3.14
```

[5]:
```
pi = 3.141592653
# Using f strings (available from Python 3.6)
print(f"{pi:.2f}")
```

```
3.14
```

[7]:
```
# area of a circle (adjusted to 4 decimal places after point)
radius = int(input())
area = 3.14 * radius * radius
print("Area is: %.4f"%area) # using % formatting (old)
print("Area is: {:.4f}".format(area)) # using .format() method on strings
print(f"Area is: {area:.4f}")
```

```
9
Area is: 254.3400
Area is: 254.3400
Area is: 254.3400
```

2 Decision Making Using Conditional Statements

- Conditional statements are used to take decisions
- There are three conditional statements in Python 1. if 2. else 3. elif

2.1 if statement

Syntax:
if condition: > block of statements

if condition is True block of statements will be executed

[6]:
```
age = int(input("Enter your age: "))#25
if age > 18: #25 > 18
    print('Yes you can vote!')
    print('Done')
```

```
Enter your age: 12
```

[9]:
```
age = int(input("Enter your age: "))#25
if age < 18: #25 > 18
    print('No you cannot vote!')
    print('Done')
```

```
Enter your age: 25
```

[12]:
```
age = int(input("Enter your age: "))#25
if age > 18: #-146 > 18
    print('Yes you can vote!')
else:
    print('No you cannot vote')
```

```
Enter your age: -146
No you cannot vote
```

[15]:
```
age = int(input("Enter your age: "))#45
if age < 18: #45 < 18
    print('No you cannot vote!')
else:
    print('You can vote!')
```

```
Enter your age: 12456
You can vote!
```

[17]:
```
# Find out the largest of two number
a = int(input())
b = int(input())
if a>b:
    print(a, 'is the largest')
else:
    print(b, 'is the largest')
```

```
100
-100
100 is the largest
```

2.2 Decisions Based on Multiple Conditions

- Usage of logical operators in conditional statements

2.2.1 Largest of three numbers

[3]:
```
a = int(input())
b = int(input())
c = int(input())
if a > b and a > c:
    print(a)
elif b > a and b > c:
    print(b)
else:
    print(c)
```

```
10
30
20
30
```

2.2.2 Result of an examination

- 5 subjects E P M CH CS
- Pass marks >=35
- Pass in all subjects

[3]:
```
E, P, M, CH, CS = map(int, input().split())
if M >= 35 and P >= 35 and CH >= 35 and E >= 35 and CS >= 35:
    print("Pass")
else:
    print("Fail")
```

```
90 90 90 90 20
Fail
```

2.2.3 Vowel or Consonant

[5]:
```
ch = input()
if ch=='a' or ch=='e' or ch=='i' or ch=='o' or ch=='u' or ch=='A' or ch=='I' or␣
 ↪ch=='E' or ch=='O' or ch=='U':
    print("Vowel")
else:
    print("Consonant")
```

```
z
Consonant
```

2.3 Multiway Conditional Check (Using elif)

- We use elif after an if statement when there are more than two possible outcomes
- elif should be preceded by an if statement and elif takes a condition just like if
- elif should not be written without a previous if
- an if statement can be followed by n number of elif statments
- it's not necessary to write an else statement after elif statment

[ ]:
```
# Check a number is even or odd --> 2 (Even, Odd)
# Find out the result of transaction based on CP and SP --> 3 (P/L/NPNL)
# Find out the largest of two given integers --> 3 (1st/2nd/BE)
# Determine if a person can cast his/her vote or not based on age -> 2 (Yes/No)
# Find out if the given number is positive or negative --> 3 (P/N/Z)
```

[8]:
```
cp = int(input()) # 100
sp = int(input()) # 100
if cp > sp: # 100 > 100
    print('Loss')
elif cp == sp: # 100 < 100
    print('NPNL')
else:
    print('Profit')
```

```
100
100
NPNL
```

[11]:
```
n = int(input())
if n > 0:
    print('Positive')
elif n == 0:
    print('Zero')
else:
    print('Negative')
```

```
0
Zero
```

[15]:
```
a, b = map(int, input().split())
if a > b:
    print('a is largest')
elif a < b:
    print('b is largest')
else:
    print('Both are equal')
```

```
20 20
Both are equal
```

2.3.1 if-elif-else ladder

[24]:
```
# print week name when week day is given
day = int(input())
if day == 1:
    print('Mon')
elif day == 2:
    print('Tue')
elif day == 3:
    print('Wed')
elif day == 4:
    print('Thu')
elif day == 5:
    print('Fri')
elif day == 6:
    print('Sat')
elif day == 7:
    print('Sun')
else:
    print('Invalid Input')
```

```
8
```

2.3.2 Conversion of a numeric grade to character grade

Let's suppose that you are given **percentage** of a students based on which you can set the grade. Assume that the grading criteria is as follows

Percentage Range	Grade
percentage>=90	O
80<=percentage<90	A
70<=percentage<80	B
60<=percentage<70	C
50<=percentage<60	D
35<=percentage<50	E
percentage<35	F

[1]:
```
per = float(input())
if per >= 90:
    print("O")
elif per >= 80 and per < 90:
    print("A")
elif per >= 70 and per < 80:
    print("B")
elif per >= 60 and per < 70:
    print("C")
elif per >= 50 and per < 60:
    print("D")
```

```
elif per >= 35 and per < 50:
    print("E")
else:
    print("F")
```

```
54.2
D
```

3 Introduction to repetition - Loops

- Doing a process again and again based on a condition to achieve a task is looping
- Python supports two different types of loops
- while loop
- for loop

3.1 while loop

- Works based on a condition
- Syntax
  while condition:
  > block of statements

block of statements will be executed **as long as the condition is true**

3.1.1 printing 1 to 10 numbers using while loop

[4]:
```
i = 1
while i <= 10: # 11 <= 10
    print(i) # 1 2 3 4 5 6 7 8 9 10
    i += 1 # i = 11
```

```
1
2
3
4
5
6
7
8
9
10
```

[5]:
```
i = 1
while i <= 10: # 11 <= 10
    print(i, end = ' ') # 1 2 3 4 5 6 7 8 9 10
    i += 1 # i = 11
```

```
1 2 3 4 5 6 7 8 9 10
```

[7]:
```
i = 1
while i <= 1000:
    print(i, end = ' ')
    i += 1
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84
85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108
109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128
129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148
149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168
169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188
189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208
209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228
229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248
249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268
269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288
289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308
309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328
329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348
349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368
369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388
389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408
409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428
429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448
449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468
469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488
489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508
509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528
529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548
549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568
569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588
589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608
609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628
629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648
649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668
669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688
689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708
709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728
729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748
749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768
769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788
789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808
809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828
829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848
849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868
869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888
```

```
889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908
909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928
929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948
949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968
969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988
989 990 991 992 993 994 995 996 997 998 999 1000
```

3.1.2 Printing numbers from 1 to n

[9]:
```
n = int(input("Enter a number: "))
i = 1
while i <= n:
    print(i, end = ' ')
    i += 1
```

```
Enter a number: 146
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84
85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108
109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128
129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146
```

3.1.3 Printing numbers from a to b

[10]:
```
a = int(input())  # 10 11 12 13 14 15
b = int(input()) # 20
while a <= b:
    print(a, end = ' ')
    a += 1
```

```
50
75
50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
```

3.1.4 Printing numbers from 10 to 1

[11]:
```
i = 10
while i >= 1:
    print(i, end = ' ')
    i -= 1
```

```
10 9 8 7 6 5 4 3 2 1
```

3.1.5 Printing numbers from n to 1

[12]:
```
n = int(input("Enter a number: "))
while n >= 1:
    print(n, end = ' ')
    n -= 1
```

```
Enter a number: 100
100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76 75
74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48
47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

[16]:
```
a = int(input())
b = int(input())
while a <= b:
    print(a, a**2, a**3)
    a += 1
```

```
2
5
2 4 8
3 9 27
4 16 64
5 25 125
```

Multiplication Table of any number upto 12
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
……
……
5 x 12 = 60

[19]:
```
i = 1
while i <= 12:
    print(f'{5} x {i} = {5 * i}')
    i += 1
```

```
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

```
5 x 11 = 55
5 x 12 = 60
```

[20]:
```
n = int(input("Which table? "))
i = 1
while i <= 12:
    print(f'{n} x {i} = {n * i}')
    i += 1
```

```
Which table? 1749
1749 x 1 = 1749
1749 x 2 = 3498
1749 x 3 = 5247
1749 x 4 = 6996
1749 x 5 = 8745
1749 x 6 = 10494
1749 x 7 = 12243
1749 x 8 = 13992
1749 x 9 = 15741
1749 x 10 = 17490
1749 x 11 = 19239
1749 x 12 = 20988
```

3.2 for loop in Python

- In Python for loop always works on iterables (list, str, range, tuple, set)
- Non iterables (int, float)
- Syntax:
  for element in iterable:
  > block of statements

[4]:
```
x = [10, 20, 30]
print(x)
print(type(x))
```

```
[10, 20, 30]
<class 'list'>
```

[5]:
```
x = [10, 20, 30]
for element in x: # element = 30
    print(element)
```

```
10
20
30
```

[6]:
```
x = [10, 20, 30]
for element in x: # element = 30
    print(element*element)
```

```
100
400
900
```

[7]:
```
x = [10, 20, 30]
for i in x:
    print(i * i)
```

```
100
400
900
```

[8]:
```
t = ('hello', 'welcome', 'to', 'python')
print(t)
print(type(t))
```

```
('hello', 'welcome', 'to', 'python')
<class 'tuple'>
```

[9]:
```
t = ('hello', 'welcome', 'to', 'python')
for word in t:
    print(word)
```

```
hello
welcome
to
python
```

[10]:
```
t = ('hello', 'welcome', 'to', 'python')
for word in t:
    print(word*3)
```

```
hellohellohello
welcomewelcomewelcome
tototo
pythonpythonpython
```

[11]:
```
s = 'hello this is python'
print(s)
print(type(s))
```

```
hello this is python
<class 'str'>
```

[12]:
```
s = 'hello this is python'
for i in s: # i = ' '
    print(i)
```

```
h
e
l
```

```
l
o

t
h
i
s

i
s

p
y
t
h
o
n
```

[14]:
```
s = 'python' # a-z -> 97-122
for i in s: # i = 'p'
    print(ord(i))
```

```
112
121
116
104
111
110
```

[13]:
```
ord('p')
```

[13]: 112

[15]:
```
n = 123
for i in n:
    print(i)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [15], in <cell line: 2>()
      1 n = 123
----> 2 for i in n:
      3     print(i)

TypeError: 'int' object is not iterable
```

[16]:
```
n = '123'
for i in n:
    print(i)
```

```
1
2
3
```

3.2.1 range() in Python

- range() is a built_in function in Python that is used to generate integers from one point to another
- range() can be used in three different ways
- range(stop)
- range(start, stop)
- range(start, stop, step)
- default:
  - start - 0
  - step - 1

range(stop)

- Generates integers from 0 to stop - 1 (In range() end bound is always excluded)
- defaults start = 0, step = 1

[20]:
```
for i in range(11): # 0 1 2 3 4 5 6 7 8 9
    print(i, end = ' ')
```

```
0 1 2 3 4 5 6 7 8 9 10
```

[21]:
```
for i in range(41):
    print(i, end = ' ')
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39 40
```

[43]:
```
for i in range(23):
    print(i, end = ' ')
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
```

range(start, stop)

- Generates integers from start to stop - 1 (In range() end bound is always excluded)
- step = 1

[22]:
```
for i in range(10, 20):
    print(i, end = ' ')
```

```
10 11 12 13 14 15 16 17 18 19
```

[23]:
```
for i in range(1, 121):
    print(i, end = ' ')
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84
85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108
109 110 111 112 113 114 115 116 117 118 119 120
```

[44]:
```
for i in range(23, 41):
    print(i, end = ' ')
```

```
23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

range(start, stop, step)

- Generate integers from start to stop - 1 (In range() end bound is always excluded) with a difference of step

[24]:
```
for i in range(10, 20, 2):
    print(i, end = ' ')
```

```
10 12 14 16 18
```

[45]:
```
for i in range(23, 46, 4):
    print(i, end = ' ')
```

```
23 27 31 35 39 43
```

[26]:
```
for i in range(100, 200, 11):
    print(i, end =  ' ') # 100 111 122 133 144 155 166 177 188 199
```

```
100 111 122 133 144 155 166 177 188 199
```

[27]:
```
for i in range(100, 199, 11):
    print(i, end =  ' ') # 100 111 122 133 144 155 166 177 188
```

```
100 111 122 133 144 155 166 177 188
```

Generating numbers backwards

- A negative step value is required
- Syntax: for element in range(start, stop, -step)

[29]:
```
for i in range(1, 11):
    print(i, end =  ' ')
```

```
1 2 3 4 5 6 7 8 9 10
```

[38]:
```
for i in range(10, 0, -1):
    print(i, end = ' ')
```

```
10 9 8 7 6 5 4 3 2 1
```

[39]:
```
for i in range(100, 50, -1): # stop - 1
    print(i, end = ' ')
```

```
100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76 75
74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51
```

[41]:
```
for i in range(1000, 200, -100):
    print(i, end = ' ')
```

```
1000 900 800 700 600 500 400 300 200
```

[42]:
```
for i in range(1000, 199, -100):
    print(i, end = ' ')
```

```
1000 900 800 700 600 500 400 300 200
```

[49]:
```
for i in range(23, 16, -8):
    print(i, end = ' ')
```

```
23
```

[50]:
```
for i in range(23, 16, -3):
    print(i, end = ' ')
```

```
23 20 17
```