Python for Data Scientists
L3: Branching & iteration

Branching & iteration

By default, statements in the script are executed sequentially from the first to the last.

Branching & iteration

The sequential flow can be altered in two ways:

- Conditional execution
- Repetitive execution

Control flow-branching

```
if <condition>:
        <expression>
        <expression>
        ….
```

```
if <condition>:
        <expression>
        <expression>
        …
else:
        <expression>
        <expression>
        …
```

```
if <condition>:
        <expression>
        <expression>
        …
elif <condition>:
        <expression>
        <expression>
        …
else:
        <expression>
        <expression>
        …
```

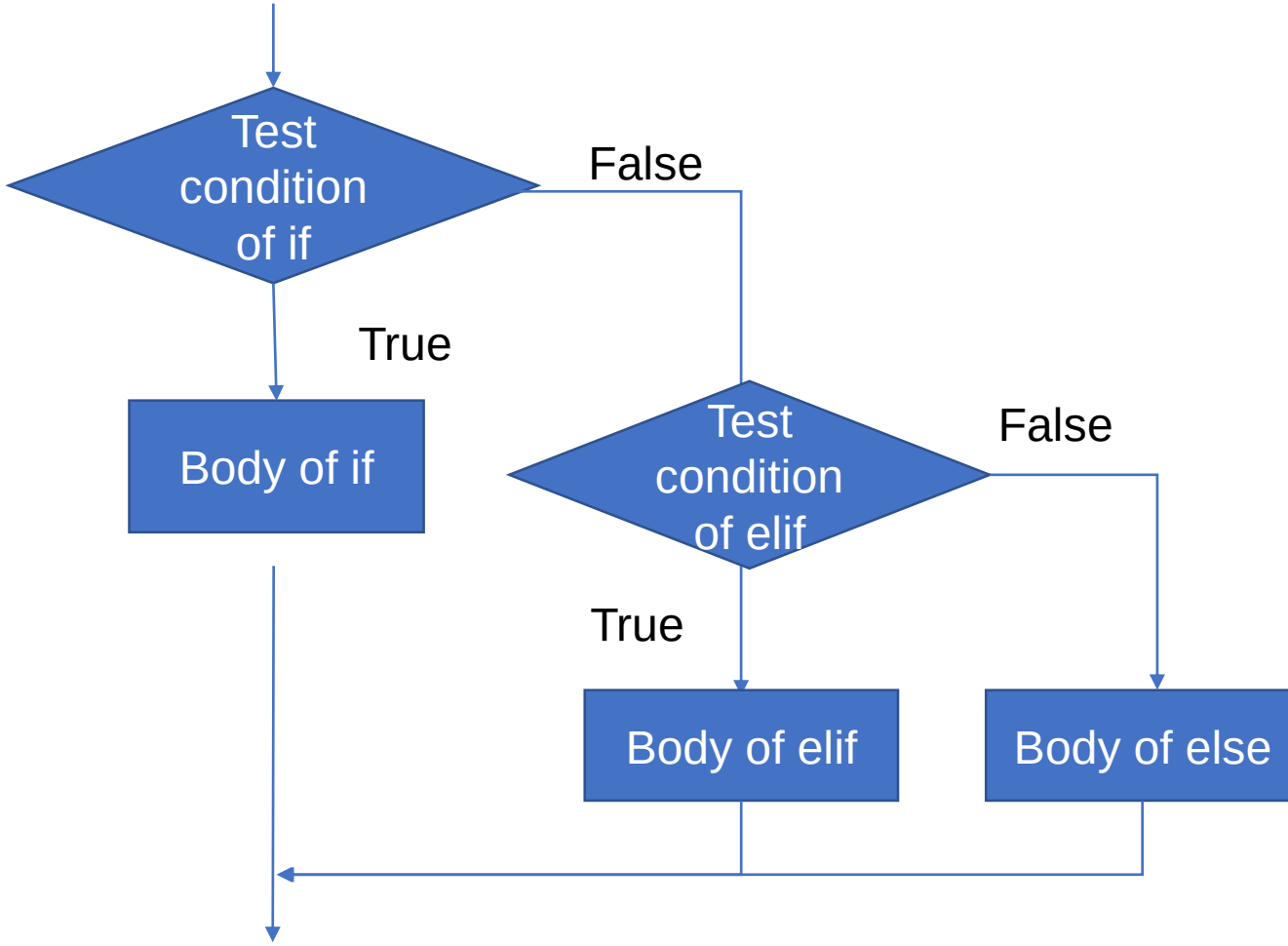
Control flow-branching

<condition> has a value True or False

- > (<) : greater (less)
- >= (<=): greater or equal (less or equal)
- == : equal
- != : not equal

Control flow-branching

- evaluate expressions in that block if <condition> is True
- the elif condition is used to include multiple conditional expressions between if and else.



Control flow-branching

Indentation in Python: helps to denote blocks of code

```
x = float(input("Enter a number for x: "))
y = float(input("Enter a number for y: "))
op = str(input("Enter a operation for op: "))

if op == '+':
    print('this is an addition', x+y)
elif op == '-':
    print('this is a subtraction’, x-y)
elif op == '*':
    print('this is a multiplication', x*y)
elif op == '/':
    if y == 0:
        print('error: division by 0')
    else:
        print('this is a division', x/y)
print('Thank you')
```

```
Enter a number for x: 2
Enter a number for y: 4
Enter a number for op: /
this is a subtraction 0.5
Thank you
```

Short Hand If

You can put if on the same line as the statement if you only have one statement to execute

```
x = int(input("Enter a number for x: "))
if x>0: print('x is a positive number')
```

Short Hand If ... Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

```
x = int(input("Enter a number for x: "))
y = int(input("Enter a number for y: "))
print("x is greater than y") if x > y else print("y is greater than x")
```

Short Hand If ... Else

You can also have multiple else statements on the same line:

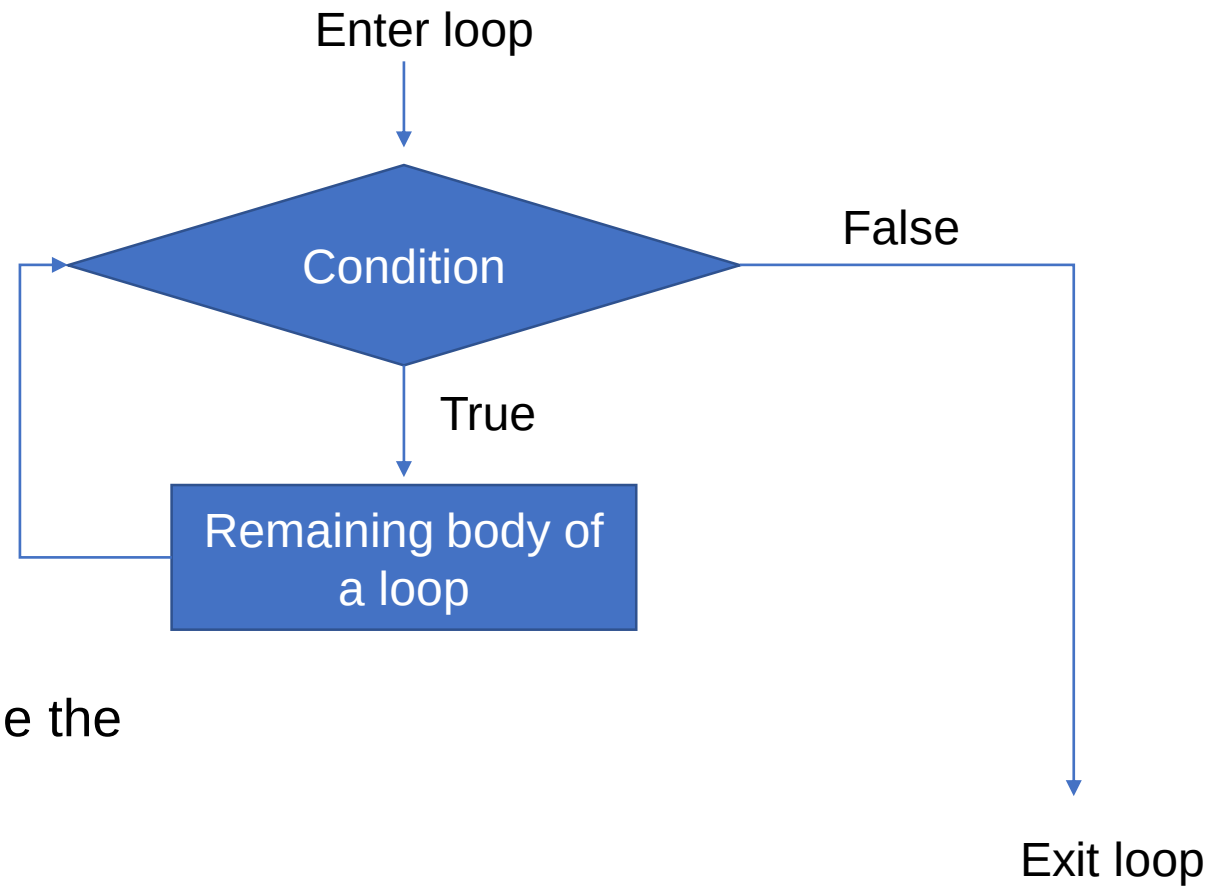
```
x = int(input("Enter a number for x: "))
y = int(input("Enter a number for y: "))
print("x is greater than y") if x > y else print("y is greater than x") if x < y else
print("y is equal to x")
```

While Loop

```
While <condition>:
        <expression>
        <expression>
        ….
```

While Loop

```
While <condition>:
        <expression>
        <expression>
        ….
```



- <condition> evaluates to a Boolean
- if <condition> is True, do all the steps inside the while code block
- check <condition> again
- repeat until <condition> is False

While Loop

```
x = int(input("Enter a number for x: "))
result=1

while x > 1:
    result = result * x
    x = x - 1
print(result)
```

```
Enter a number for x: 4
24
```

While loop (Poll)

What will be the output of this code:

```
d = {'one': 1, 'two': 2, 'three': 3}
while d:
    print(d.popitem())
print('Done.')
```

- A: Done.
- B: ('three', 3)
  ('two', 2)
  ('one', 1)
  Done.
- C ['three’, 3]
  ['two’, 2]
  ['one’, 1]
  Done.

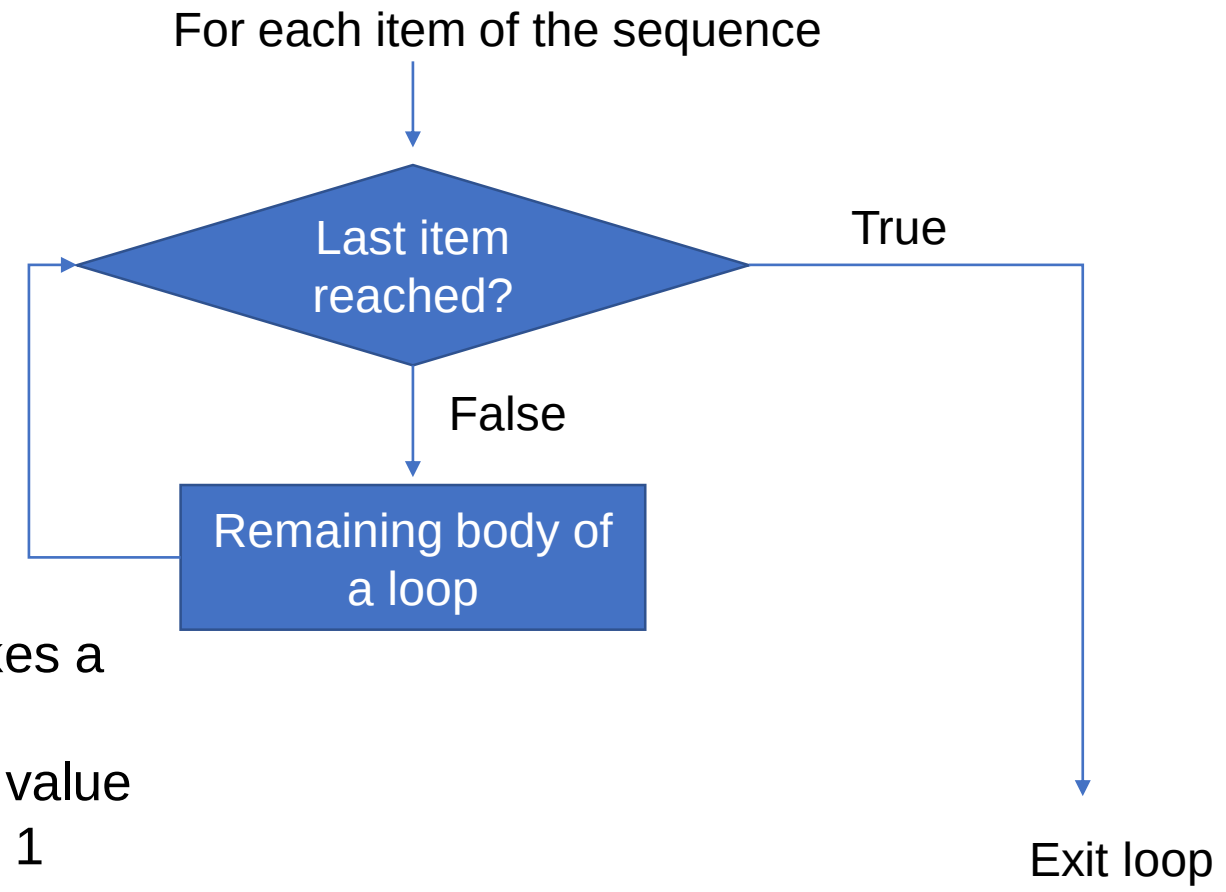
The .popitem() method removes one key-value pair from d and returns it as a tuple

For Loop

```
for  <variable> in range (<someNumber>):
        <expression>
        <expression>
        ….
```

For Loop

```
for  <variable> in range (<someNumber>):
            <expression>
            <expression>
            ….
```



- each time through the loop, <variable> takes a value
- first time, <variable> starts at the smallest value
- next time, <variable> gets the prevvalue + 1
- range(start,stop,step)

For Loop (Poll)

```
sum= 0

for i in range(0, 10, 3):
    sum+= i

print(sum)
```

- Start :0
- Stop: 10
- Step : 3

A	B	C
45	18	55

Infinite loop

Loop that runs forever and can be stopped only by killing the program or restarting the computer:

- forgetting to update the variable that controls the loop

```
i = 0
while i <= 10:
    print(i)
```

- accidentally incrementing a counter that should be decremented

Nested for Loop

- If a loop (for loop or while loop) contains another loop in its body block, we say that the two loops are nested.

- If the outer loop is designed to perform m iterations and the inner loop is designed to perform n repetitions, the body block of the inner loop will get executed m X n times.

Loop over String

```
word='Python'
index = 0
while index < len(word):
    letter = word[index]
    print (letter)
    index = index + 1
```

```
for char in word:
    print (char)
```

Loop over List

```
for i in range(len(numbers)):
    print(numbers[i])
```

```
for num in numbers:
    print (num)
```

Loop over tuple

```
tup = ((1, "chalmers"), (4, "university"), (9, "python"), (1, "course"))

words = ()
for t in tup:
    if t[0] == 1:
        words = words + (t[1],)
        unique_words = len(words)

print(unique_words,' : ',words)
```

```
2 : ('chalmers', 'course')
```

Loop over dictionary

```
dict={ 1:100, 2:200, 3:300 }
for pair in dict.items():
    print (pair)
```

```
(1, 100)
(2, 200)
(3, 300)
```

```
dict={ 1:100, 2:200, 3:300 }
for k,v in dict.items():
    print("key=", k, ", value=", v)
```

```
key= 1 , value= 100
key= 2 , value= 200
key= 3 , value= 300
```

```
dict={1:100, 2:200, 3:300}
for k in dict.keys():
    print(k, dict.get(k))
```

```
1 100
2 200
3 300
```

Else in a Loop

- Python allows the else keyword to be used with the for and while loops too.
- The else block appears after the body of the loop.
- The statements in the else block will be executed after all iterations are completed.

Else in a Loop

```
for x in range(3):
    print ("iteration num",x+1, " in for loop")
else:
    print ("else block in loop")
print ("Out of loop")
```

```
iteration num 1  in for loop
iteration num 2  in for loop
iteration num 3  in for loop
else block in loop
Out of loop
```

Else in a Loop

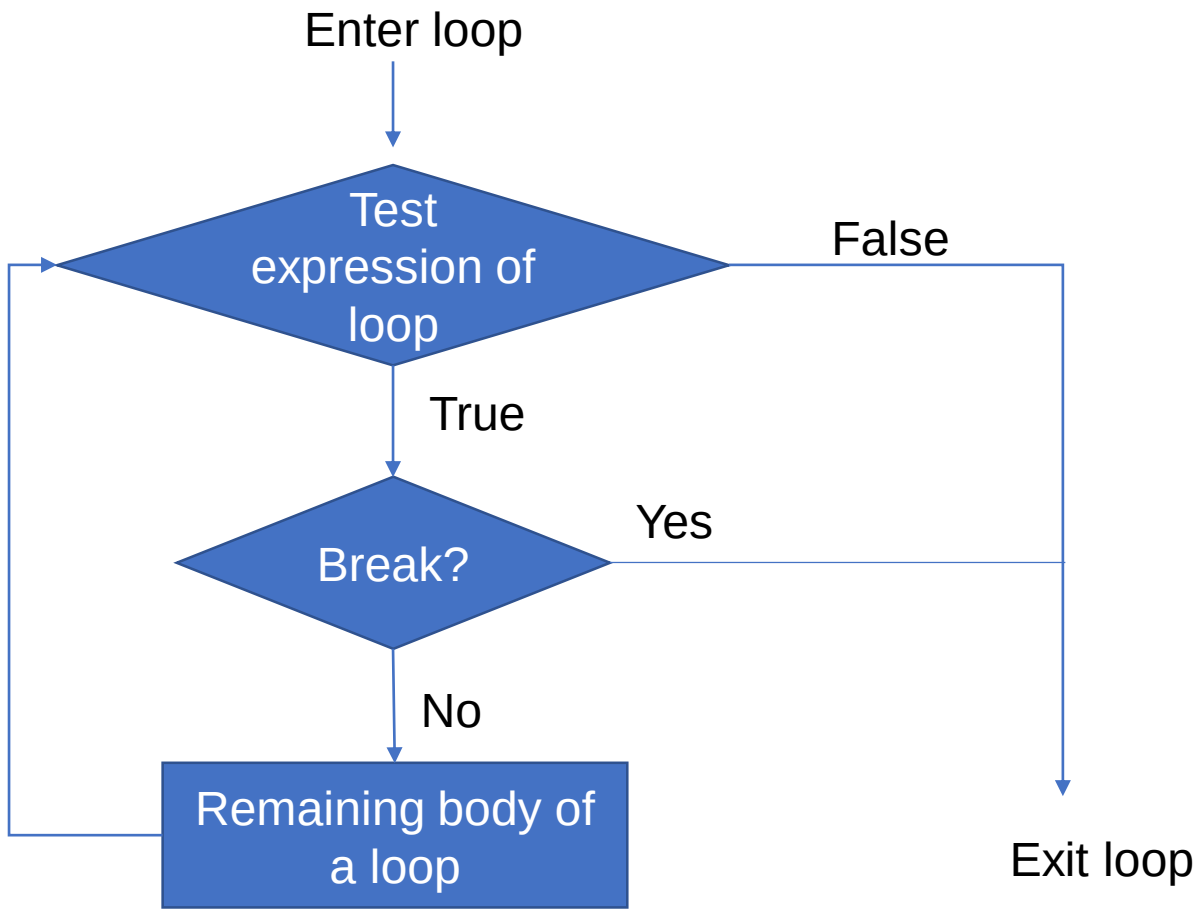
```
x=0
while x<3:
    x=x+1
    print ("iteration num",x, " in while loop")
else:
    print ("else block in loop")
print ("Out of loop")
```

```
iteration num 1  in while loop
iteration num 2  in while loop
iteration num 3  in while loop
else block in loop
Out of loop
```

Break statement in a loop

- immediately exits whatever loop it is in
- skips remaining expressions in code block
- exits only innermost loop!

Break statement in a loop



Break statement in a loop (Poll)

```
for num in range(1,10):
    print ("Num = ", num)
    if num==4:
        break
print ("Out of loop")
```

```
Num =  1
Num =  2
Num =  3
Num =  4
Out of loop
```

What will be the last number to be printed?

A	B	C
4	9	10

Break statement in a loop (Poll)

```
for i in range(1, 4):
    for j in range(1,5):
        print('Hello')
        break
```

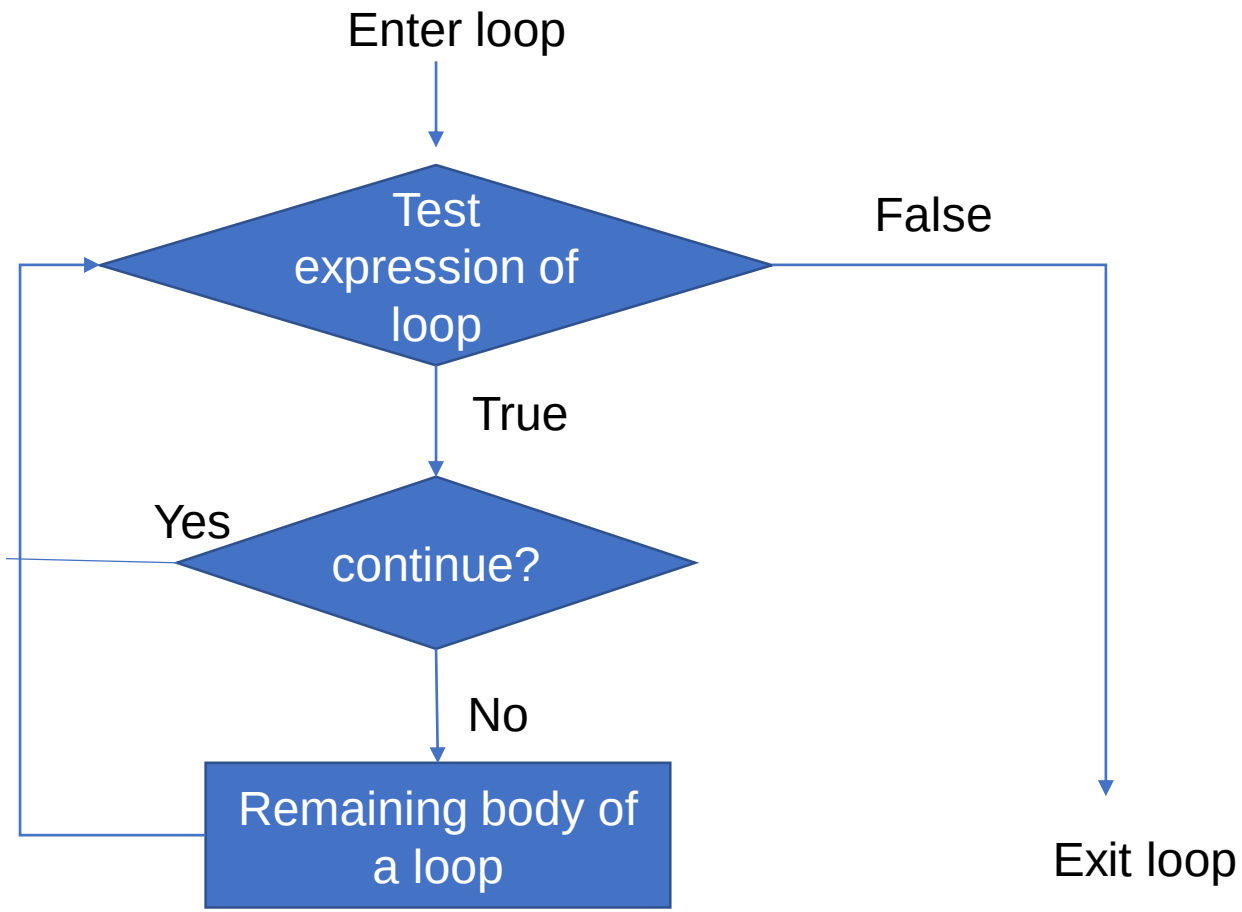
How many time ‘hello’ will be printed?

A	B	C
3	1	12

Continue statement in a loop

- The continue statement is somewhat opposite to the break.
- It skips the remaining statements in the current loop and starts the next iteration.

Continue statement in a loop



Continue statement in a loop

```
num=0
while num<6:
    num=num+1
    if num==4:
        continue
    print ("Num : ", num)
print ("Out of loop")
```

```
Num :  1
Num :  2
Num :  3
Num :  5
Num :  6
Out of loop
```

Continue statement in a loop (Poll)

```
num=int(input("enter a number"))
d=2
while num>1:
    if num % d==0:
        print (d)
        num=num/d
        continue
    d=d+1
```

continue causes the division of the input number by the same value of the divisor repetitively. The next value of the divisor is taken only after the division is not possible.

enter a number30

2

3

5

what happens in this program and what will be the last printed value of d if num = 30?

A	B	C
5	3	2

Pass statement

- The pass is used as a dummy place holder whenever a syntactical requirement of a certain programming element is to be fulfilled without assigning any operation.

- Python interpreter ignores pass

Pass statement

```
for num in range(1,6):
    if num==4:
        pass
    else:
        print ("Num : ", num)
```

```
Num : 1
Num : 2
Num : 3
Num : 5
```

For vs While Loops

- For loops
  - **Know** number of iterations
  - can **end early** via break
  - uses a **counter**
  - **Can rewrite** a for loop using a while loop

- While loops
  - **Unbounded** number of iterations
  - can **end early** via break
  - can use a **counter but must initialize** before loop and increment it inside loop
  - **may not be able to rewrite** a while loop using a for loop

Looping techniques in Python

- Python supports various looping techniques by certain inbuilt functions, in various sequential containers.

- These methods are primarily very useful in competitive programming and in various project which require a specific technique with loops maintaining the overall structure of code.

Looping techniques in Python

Where they are used ?

- useful in the places where we don’t need to actually manipulate the structure and ordering of overall container

- Useful where we only need to print the elements for a single use instance, no inplace change occurs in the container.

Looping techniques in Python: enumerate()

enumerate(): loop through the containers printing the index number along with the value present in that particular index.

```
for key, value in enumerate(['Python', 'for', 'data', 'scientist']):
    print(key, value)
```

```
0 Python
1 for
2 data
3 scientist
```

Looping techniques in Python: zip()

zip(): combine 2 similar containers(list-list or dict-dict) printing the values sequentially. The loop exists only till the smaller container ends.

```
questions = ['name', 'color', 'shape']
answers = ['apple', 'red', 'a circle']

for question, answer in zip(questions, answers):
    print('What is your {0}? I am {1}.'.format(question, answer))
```

```
What is your name? I am apple.
What is your color? I am red.
What is your shape? I am a circle.
```

Looping techniques in Python: items()

items(): loop through the dictionary printing the dictionary key-value pair sequentially.

```
d = {"un": "one", "deux": "two"}

print("The key value pair using items is : ")
for i, j in d.items():
    print(i, j )
```

```
The key value pair using items is :
un one
deux two
```

Looping techniques in Python: sorted()

sorted(): print the container in a sorted order without sorting the original list.

```
l = [0, 2, 4, 7, 0, 1, 4]


print("The list in sorted order is : ")
for i in sorted(l):
    print(i, end=" ")
```

```
The list in sorted order is :
0 0 1 2 4 4 7
```

Looping techniques in Python: sorted() & set()

set() can be combined with sorted() to remove duplicate occurrences.

```
l = [0, 2, 4, 7, 0, 1, 4]

print("The list in sorted order (without duplicates) is : ")
for i in sorted(set(l)):
    print(i, end=" ")
```

```
The list in sorted order (without duplicates) is :
0 1 2 4 7
```

Looping techniques in Python: reversed()

reversed(): print the values of container in the reversed order without doing any changes to the original list

```
l = [0, 2, 4, 7, 0, 1, 4]

print ("The list in reversed order is : ")
for i in reversed(l) :
   print (i,end=" ")
```

The list in reversed order is :
4 1 0 7 4 2 0

Switch-case statement

- It allows to control the flow of a program based on the value of a variable or an expression

- How it works:
  - Compiler generates a jump table for switch case statement
  - The switch variable/expression is evaluated once
  - Switch statement looks up the evaluated variable/expression in the jump table and directly decides which code block to execute.
  - If no match is found, then the code under default case is executed

Python does not have a switch or case statement!

```
switch (month) {
        case 1:  monthString = "January";
                 break;
        case 2:  monthString = "February";
                 break;
        case 3:  monthString = "March";
                 break;
        case 4:  monthString = "April";
                 break;
        case 5:  monthString = "May";
                 break;
        case 6:  monthString = "June";
                 break;
        case 7:  monthString = "July";
                 break;
        case 8:  monthString = "August";
                 break;
        case 9:  monthString = "September";
                 break;
        case 10: monthString = "October";
                 break;
        case 11: monthString = "November";
                 break;
        case 12: monthString = "December";
                 break;
        default: monthString = "Invalid month";
                 break;
    }
```

Switch-case statement

Solutions:

- use a series of if-else-if blocks

- Implement switch statement by using a dictionary mapping

Switch-case statement

```
switcher = {
        1: "January",
        2: "February",
        3: "March",
        4: "April",
        5: "May",
        6: "June",
        7: "July",
        8: "August",
        9: "September",
        10: "October",
        11: "November",
        12: "December"
    }

print (switcher.get(2, "Invalid month"))
print (switcher.get(19, "Invalid month"))
```

- Look up against the switcher dictionary mapping.
- If a match is found, the associated value is printed, else a default string (‘Invalid month’) is printed.
- The default string helps implement the ‘default case’ of a switch statement.