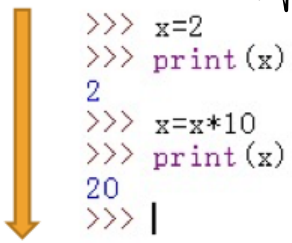
Lecture 3 Flow Control (2021.9.23 & 26)

§1 Transfer of Control

1. Sequential Flow

Statements are executed one after another in their written order.



```
>>> x=2
>>> print(x)
2
>>> x=x*10
>>> print(x)
20
>>>
```

2. Transfer of Control

- Some steps are conditional
- Some steps should be repeated

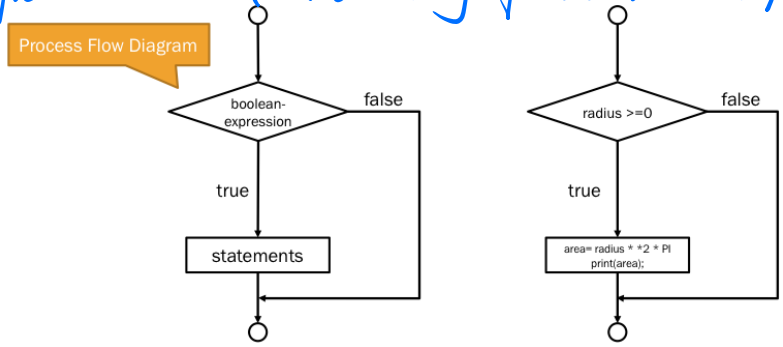
3. Structured Programming

1° There are only three control structures

- sequence structure: sequential flow (default structure)
- selection structure: conditional flow (if, if/else, elif)
- repetition structure: repeated flow (while, for)

§2 Selection Structure

1. Single Selection (One-way if statement)



2. Boolean Type

1° Conditional flow need a Yes/No result. Officially named True/False.

2° bool( )

```
>>> x = 0; y = 0.0; z = 0 + 0j
>>> bool(x), bool(y), bool(z)
(False, False, False)
>>> x = -1; y = 1.e-10; z = 0 + 1j
>>> bool(x), bool(y), bool(z)
(True, True, True)
>>> x = []; y = [0]; z = "0"
>>> bool(x), bool(y), bool(z)
(False, True, True)
```

Imaginary Part

Exponent

3. Comparison Operators

1° Boolean Expressions

produce Boolean values

use comparison operators to evaluate

2° Comparison Operators

check variables

do not change the values of variables

x < y	Is x less than y?
x <= y	Is x less than or equal to y?
x == y	Is x equal to y?
x >= y	Is x greater than or equal to y?
x > y	Is x greater than y?
x != y	Is x not equal to y?

```
>>> 5 > 7                # Is 5 greater than 7?
False
>>> x, y = 45, -3.0
>>> x > y                # Is 45 greater than -3.0?
True
>>> result = x > y + 50 # Is 45 greater than -3.0 + 50?
>>> result
False
>>> "hello" > "Bye"      # Comparison of strings.
True
>>> "AAB" > "AAC"
False

>>> 7 == 7.0
True
>>> x = 0.1
>>> 1 == 10 * x
True
>>> 1 == x + x + x + x + x + x + x + x + x + x
False
>>> x + x + x + x + x + x + x + x + x + x
0.9999999999999999
>>> 7 != "7"
True
>>> 'A' == 65
False
```

ord('A') == 65

'A' == chr(65)

4. Indentation

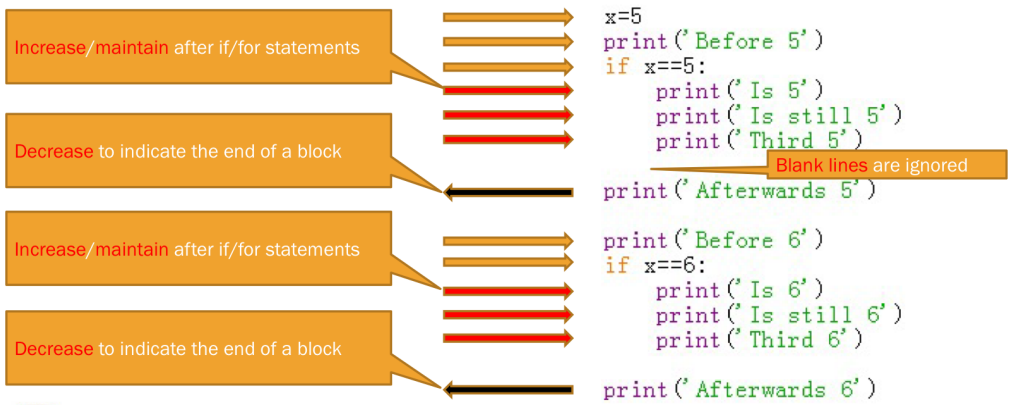
1° Increase indent: indent after an if statement (after :)

2° Maintain indent: to indicate the scope of the block (which lines are affected by the if/for)

3° Decrease indent: to back to the level of the if statement or for statement to indicate the end of the block

4° Blank lines are ignored — they do not affect indentation

5° Comments on a line by themselves are ignored with respect to indentation.




```
x=5
if x==5:
    print("Equals 5")

if x>4:
    print("Greater than 4")

if x>=5:
    print("Greater than or equal to 5")

if x<=5:
    print("Less than or equal 5")

if x!=6:
    print("Not equal 6")
```

```
Equals 5
Greater than 4
Greater than or equal to 5
Less than or equal 5
Not equal 6
```

```
x=5
print('Before 5')
if x==5:
    print('Is 5')
    print('Is still 5')
    print('Third 5')

print('Afterwards 5')

print('Before 6')
if x==6:
    print('Is 6')
    print('Is still 6')
    print('Third 6')

print('Afterwards 6')
```

```
Before 5
Is 5
Is still 5
Third 5
Afterwards 5
Before 6
Afterwards 6
```

5. Nested Decisions

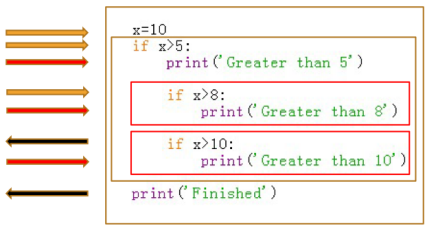
1. Mental Begin/End

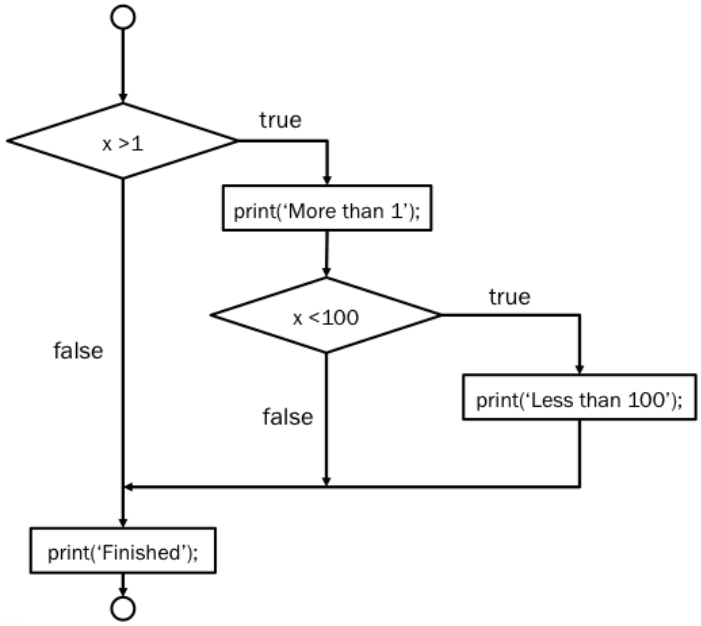
Example

```
x=42
if x>1:
    print('More than 1')

    if x<100:
        print('Less then 100')

print('Finished')
```

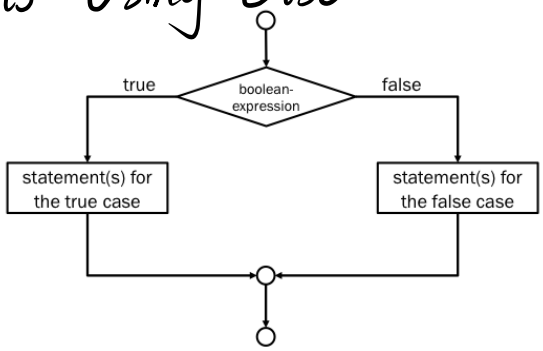




b. Conditional Flow

1° Two Way Decisions Using Else

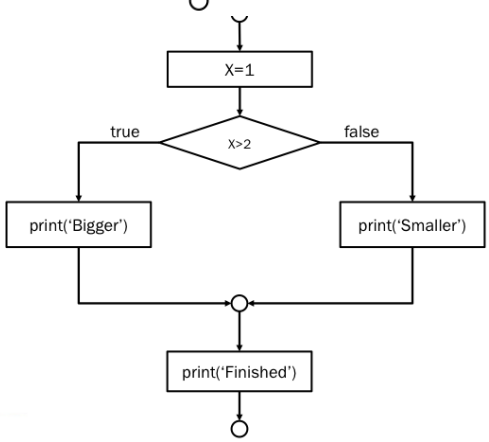
- Sometimes we want to do one thing when the logical expression is true, and another thing when it is false
- It is like a fork in the road, we need to choose one or the other path, but not both




```
x=1

if x>2:
    print('Bigger')
else:
    print('Smaller')

print('Finished')
```



2° Tips On If-Else

① else must come after if

② Use indentation to match if and else.

3° Multi-way Decisions

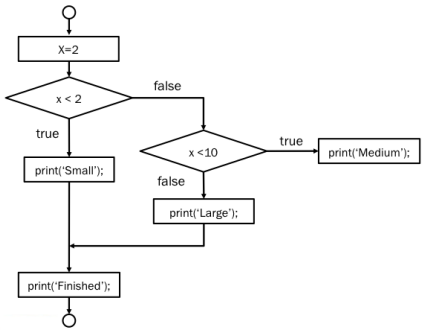
```
#No else

x=2
if x<2:
    print('Small')
elif x<10:
    print('Medium')

print('Finished')
```

```
x=2
if x<2:
    print('Small')
elif x<10:
    print('Medium')
else:
    print('Large')

print('Finished')
```



§3 Logical Operators

1. What's Logical Operators

1° Be used to combine several logical expressions into a single expression

2° Python has three logical operators: not , and , or

3° Highest to lowest precedence rule

1. Parenthesis
2. Power
3. Multiplication, Division and Remainder
4. Addition and Subtraction
5. not
6. and
7. or
8. if-else
9. Left to Right

```
>>> not True
False
>>> False and True
False
>>> not False and True
True
>>> (not False) and True
True
>>> True or False
True
```

```
>>> not False or True        # Same as: (not False) or True.
True
>>> not (False or True)
False
>>> False and False or True  # Same as: (False and False) or True.
True
>>> False and (False or True)
False
```

§4 Try / Except Structure

1. What's Try/Except Structure

Surround a dangerous part of code with try/except

1° If the code in try block fails, the except block will be executed

2° If the code in try block works, the except block will be skipped

```
astr = 'Hello bob'
try:
    istr = int(astr)
except:
    istr = -1
print('First', istr)

astr = '123'
try:
    istr = int(astr)
except:
    istr = -1
print('Second', istr)
```

Fail: stops into the except block, and the program continues

Succeed: skips the except block

```
→ code python3 err.py
First -1
Second 123
→ code
```

EXAMPLE

```
rawstr = input('Enter a number:')

try:
    ival = int(rawstr)
except:
    ival = -1

if ival>0:
    print('Nice work')
else:
    print('Invalid number')
```

§5 Repetition Structure

1. Repeated Flow

Loops (repeated steps) have iterative variables that change each time through a loop

Often these iterative variables go through a sequence of numbers

2. Comparison Between Indefinite loop (While loop) and Definite Loop (For loop)

1° Indefinite loop:

① While loops are called indefinite loops

② Keep going until a logical condition becomes false

③ It can be hard to determine whether a loop will terminate

2° Definite loop:

① Quite often we have a finite set of items

② Each iteration of which will be executed for each item in the set.

③ Use the for statement

Program

```
n=5
while n>0:
    print(n)
    n = n - 1
print("Finish")
```

Outputs

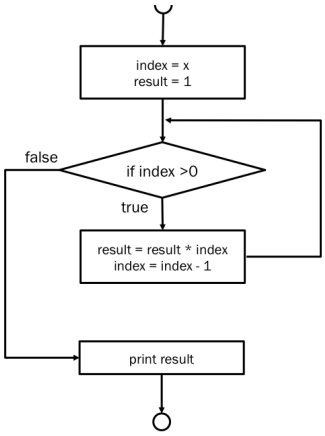
```
5
4
3
2
1
Finish
>>>
```

factorial.py

```
index = int(input("Please input the data:"))
result = 1

while index>0:
        result = result*index
        index -= 1

print(result)
```



Example

Iteration variable

Ordered Sequence with five elements

The iteration variable moves through all of the values in the sequence

```
for i in [5, 4, 3, 2, 1]:
    print(i)
print('Finished')
```

The block (body) of the code is executed once for each value in the sequence

Output

```
5
4
3
2
1
Finished
```

Example

```
friends = ['Tom', 'Jerry', 'Bat']
for friend in friends:
    print('Happy new year', friend)
print('Done')
```

Output

```
Happy new year Tom
Happy new year Jerry
Happy new year Bat
Done
```

Note: though these examples are simple, the patterns apply to all kinds of loops

3. Making "Smart" For Loops

For variables:

1° Set some variables to initial values

For thing in data:

2° Look for something or do something to each entry separately, updating a variable.

3° Look at the variables.

COUNTING IN A LOOP

- To count how many times we have executed a loop, we can introduce a counting variable, which increases itself in each iteration

Example

```
count = 0
print('Before', count)
for thing in [3, 4, 98, 38, 9, 10, 199, 78]:
    count = count + 1
    print(count, thing)
print('After', count)
```

Output

```
Before 0
1 3
2 4
3 98
4 38
5 9
6 10
7 199
8 78
After 8
```

FILTERING IN A LOOP

- We can use an if statement in a loop to catch/filter the values we are interested at

Example

```
print('Before')

for value in [23, 3, 43, 39, 80, 111, 99, 3, 65]:
    if value>50:
        print('Large value:', value)

print('After')
```

Output

```
Before
Large value: 80
Large value: 111
Large value: 99
Large value: 65
After
```

SEARCH USING A BOOLEAN VARIABLE

- If we want to search in a set and double check whether a specific number is in that set
- We can use a Boolean variable, set it to False at the beginning, and assign True to it as long as the target number is found

Example

```
found = False

print('Before', found)

for value in [9, 41, 12, 3, 74, 15]:
    if value == 74:
        found = True
    print(found, value)
print('After', found)
```

Output

```
Before False
False 9
False 41
False 12
False 3
True 74
True 15
After True
```

FINDING THE SMALLEST NUMBER

Example

```
smallest_so_far = None
print('Before', smallest_so_far)

for num in [9, 39, 21, 98, 4, 5, 100, 65]:
    if smallest_so_far == None:
        smallest_so_far = num
    elif num < smallest_so_far:
        smallest_so_far = num
    print(smallest_so_far, num)

print('After', smallest_so_far)
```

Output

```
Before None
9 9
9 39
9 21
9 98
4 4
4 5
4 100
4 65
After 4
```

- We still use a variable to store the smallest value seen so far
- The original smallest value is None, we use an if statement to check this

NESTED LOOP

- What would be the output?

sum.py

```
number = 9
i = 1
while i<=number:
        sum = 0
        j = 1
        while j<=i:
                sum += j
                j += 1
        print("Sum of 1 to", i, "=", sum)
        i+=1
```

Output:

```
Sum of 1 to 1 = 1
Sum of 1 to 2 = 3
Sum of 1 to 3 = 6
Sum of 1 to 4 = 10
Sum of 1 to 5 = 15
Sum of 1 to 6 = 21
Sum of 1 to 7 = 28
Sum of 1 to 8 = 36
Sum of 1 to 9 = 45
```

4. The Is and Is Not Operator

1° "is" implies "is the same as"

2° Similar to, but stronger than ==

3° "is not" is also an operator.

```
smallest_so_far = None
print('Before', smallest_so_far)

for num in [9, 39, 21, 98, 4, 5, 100, 65]:
    if smallest_so_far is None:
        smallest_so_far = num
    elif num < smallest_so_far:
        smallest_so_far = num
    print(smallest_so_far, num)

print('After', smallest_so_far)
```

Example

```
print(10 is 10)

a = 10
b = 10
print (a is b)

a = '123'
b = '123'
print (a is b)

a = [1, 2, 3]
b = [1, 2, 3]
print (a is b)
```

Output

```
True
True
True
False
```

5. Range

testrange1.py

```
for x in range(6):
    print(x)
```

Output:

```
0
1
2
3
4
5
```

testrange2.py

```
for x in range(2, 6):
    print(x)
```

Output:

```
2
3
4
5
```

testrange3.py

```
for x in range(2, 30, 3):
    print(x)
```

start, end, step

Output:

```
2
5
8
11
14
17
20
23
26
29
```

testrange.py

```
for x in range(6):
    print(x)
else:
    print("Finally finished!")
```

specifies a block of code to be executed when the loop is finished

Output:

```
0
1
2
3
4
5
Finally finished!
```

6. Breaking Out of a Loop

1° The break statement ends the current loop

2° Jump to the statement which directly follows the loop

```
while (True):
    line = input('Enter a word:')
    if line == 'done':
        break
    print(line)
print('Finished')
```

7. Finishing an Iteration with Continue

The continue statement ends the current iteration, and start the next iteration immediately

```
while True:
    line = input('Input a word:')
    if line[0] == '#': continue
    if line == 'done':
        break
    print(line)
print('Done')
```

bcnested1.py

```
while condition1:
    while condition2:
        # code 1
        break
        # code 2
    # code 3
# code 4
```

bcnested2.py

```
while condition1:
    while condition2:
        # code 1
        continue
        # code 2
    # code 3
# code 4
```