Programing in Python

Lecture 3 - Conditionals and Loops

Instructor: Zhandos Yessenbayev

Outline

- Conditionals
- Loops
- Useful loop examples

Boolean expressions

- **Boolean expression** is either true or false.

```
>>> 5 == 5
True
>>> 5 == 6
False
```

- **Comparison** operators

```
x == y                  # x is equal to y
x != y                  # x is not equal to y
x > y                   # x is greater than y
x < y                   # x is less than y
x >= y                  # x is greater than or equal to y
x <= y                  # x is less than or equal to y
x is y                  # x is the same as y
x is not y              # x is not the same as y
```

Logical operators

- There are three logical operators: **and, or, not**.

```
>>> x = 1
>>> (x<0) or (x>1)
>>> (x>0) and (x<1)
>>> not (x<0)
>>> (x<0) and (x==1)
```

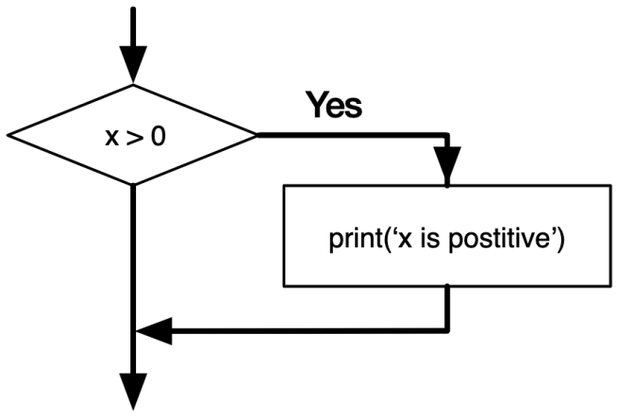
- Any nonzero number is interpreted as “true.”

```
>>> 17 and True
True
>>> False and 17
False
```

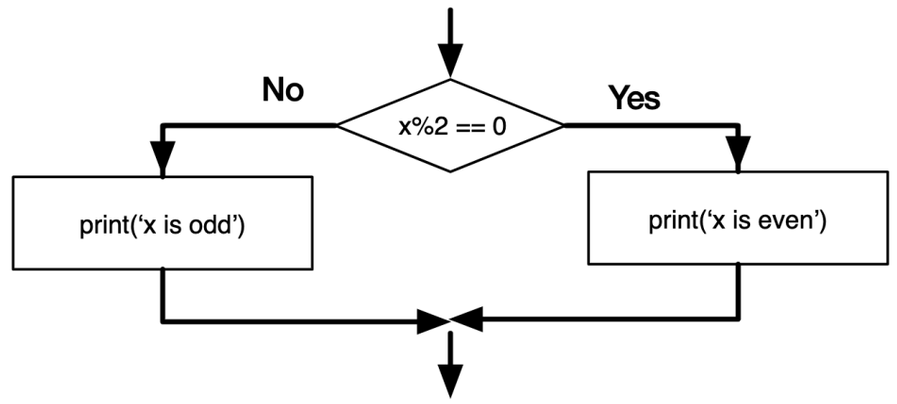
- Python evaluates the expression *from left to right*

Conditional execution

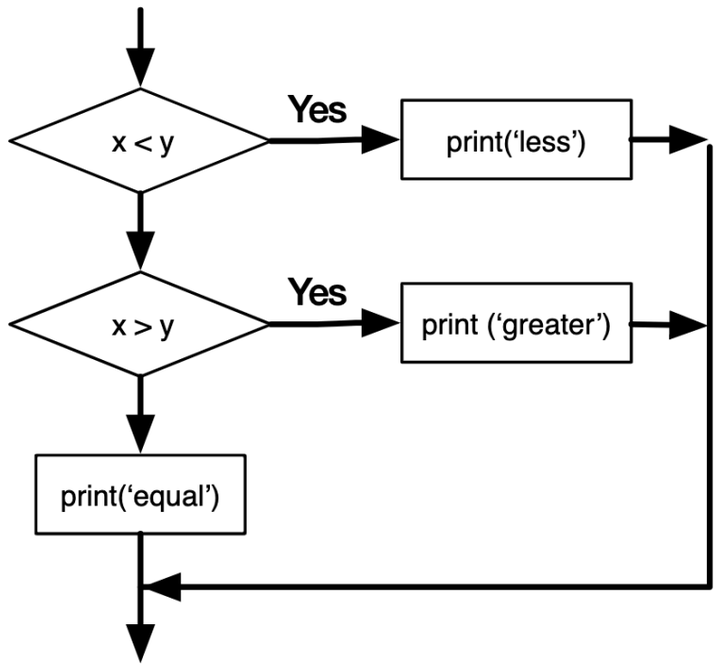
```
if x > 0 :
    print('x is positive')
```



```
if x%2 == 0 :
    print('x is even')
else :
    print('x is odd')
```

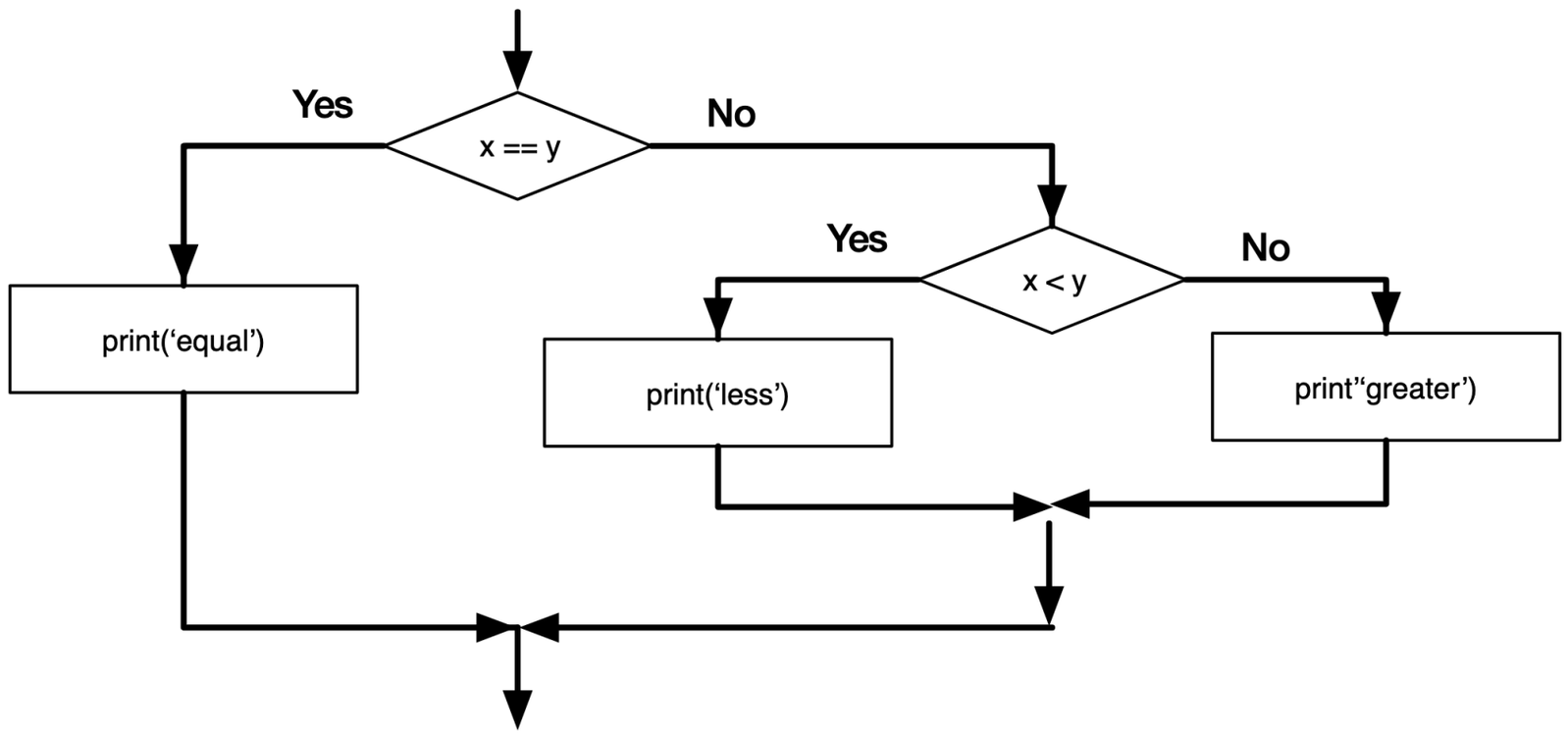


```
if x < y:
    print('x is less than y')
elif x > y:
    print('x is greater than y')
else:
    print('x and y are equal')
```



Nested Conditions

```
if x == y:
    print('x and y are equal')
else:
    if x < y:
        print('x is less than y')
    else:
        print('x is greater than y')
```



Catch Exceptions

- Programs can raise **exceptions** (errors)

```
>>> prompt = input(“Enter a number: ”)
>>> n = input(prompt)
>>> int(n)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: ''
```

- To catch and process exceptions, we use **try…except**

```
>>> try:
>>>     int(n)
>>> except:
>>>     print('Please enter a number')
```

Updating variables

- Remember assignment operator assigns some value to a variable — Variable = Value :

```
>>> x = 1
>>> x = 2
>>> x = 3
```

- Very often we will be updating variables like so:

```
>>> x = x + 1
```

- In this case, initial value must be provided to x:

```
>>> x = 1
>>> x = x + 1
```

- Otherwise we get an error:

```
>>> x = x + 1
NameError: name 'x' is not defined
```

While loop

- To perform repetitive tasks, we use **while** loop

```
>>> n = 5
>>> while n > 0:
...     print(n)
...     n = n - 1
...
>>> print(“Finally, n =”,n)
```

- General syntax:

```
while <boolean expression>:
    <loop body>
```

Infinite loop

- Try this:

```
n = 5
while 1:
    print(n)
    n = n - 1
print(“Finally, n =”,n)
```

- We get infinite loops, if *boolean expression* is always True

continue or break

- continue — skip this iteration and proceed to next one
- break — exit this loop

```
while True:
    num = input(‘Enter a number: ‘)
    if num == '0':
        continue
    if num == ‘-1':
        break
    print(num)
print('Done!')
```

Lists

- **Lists** — a special data structure in Python that stores any objects as an array.

- Examples:
  - [ ] — empty list
  - [1, 2, 3, 4, 5] — list of numbers
  - [‘Tom’, ‘Bob’, ‘Sam’] — list of strings
  - [1, 2, ‘Tom’, True, False] — list of mixed objects

For loop

- **For** loop also is used for repetitive tasks:

```
friends = ['Tom', 'Bob', ‘Sam']
for friend in friends:
    print('Happy New Year:', friend)
print(‘Done!’)

for num in [1, 2, 3, 4, 5]:
    print(‘Line:’, num)
```

- General syntax:

```
for <iterator_variable> in <some_list>:
    <loop body>
```

Loop patterns

- Loops are generally constructed by:
    - Initializing some variables before the loop starts
    - Performing some computation on each item in the loop body
    - Looking at the resulting variables when the loop completes

1. Counting loop

- Count some objects

```
count = 0
for item in [3, 41, 12, 9, 74, 15]:
    count = count + 1
print('Count: ', count)
```

2. Summing loop

- Sum the objects in the list

```
total = 0
for item in [3, 41, 12, 9, 74, 15]:
    total = total + item
print('Total: ', total)
```

3. Maximum loop

- Find maximum element in the list

```
Max = None
print('Before:', Max)

for item in [3, 41, 12, 9, 74, 15]:
    if Max is None or item > Max :
        Max = item

print('After:', Max)
```

4. Minimum loop

- Find minimum element in the list:

```
Min = None
print('Before:', Min)

for item in [3, 41, 12, 9, 74, 15]:
    if Min is None or item < Min :
        Min = item

print('After:', Min)
```

Thanks!