

Control Statements, Lists

WTP 2010
Computer Science
Day 3: Wednesday, June 30

Beyond sequential execution

So far, all our programs have looked like this:

```
<do thing 1>
<do thing 2>
<do thing 3>
...
```

**Start with first command.
Execute commands in order
until there are no more.**

But often sequential execution is not enough.

```
if <something>:
    <do thing 1>
else:
    <do thing 2>
```

**If something is true, execute
the first command. Otherwise,
execute the second command.**

2

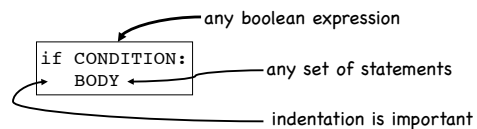
Control statements

Affect how other statements are executed.

- Conditionals: control which set of statements is executed.
 - if / else
- Iteration: control how many times a set of statements is executed.
 - while loops
 - for loops

3

The if statement



- If the CONDITION is True, the BODY gets executed.
- Otherwise, nothing happens.

```
if x < 0:
    print 'x is negative'
```

- NOTE: IDLE editor helps with indentation.

4

The if / else statement

```
if CONDITION:
    BODY1
else:
    BODY2
```

any set of statements

- If the CONDITION is True, BODY1 gets executed.
- Otherwise, BODY2 gets executed.

```
if x < 0:
    print 'x is negative'
else:
    print 'x is positive or zero'
```

5

Chained conditionals

```
if CONDITION1:
    BODY1
elif CONDITION2:
    BODY2
else:
    BODY3
```

another boolean expression

any set of statements

- If the CONDITION1 is True, BODY1 gets executed.
- Otherwise, if CONDITION2 is True, BODY2 gets executed.
- If neither condition is True, BODY3 gets executed.

6

Chained conditionals

```
if x < 0:
    print 'x is negative'
elif x > 0:
    print 'x is positive'
else:
    print 'x must be zero!'
```

7

An example

```
a = False
b = True
if a and b:
    print 'I love red.'
elif a or b:
    print 'I love green.'
else:
    print 'I love blue.'
    print 'I also love purple.'
```

What does this output?

8

An example

```
a = False
b = True
if a and b:
    print 'I love red.'
elif a or b:
    print 'I love green.'
else:
    print 'I love blue.'
    print 'I also love purple.'
```

What does this output?

9

Nested conditionals

```
if is_adult:
    if is_senior_citizen:
        print 'Admission $2 off.'
    else:
        print 'Full price.'
else:
    print 'Admission $5 off.'
```

outer conditional
inner conditional

Can get confusing. Indentation helps to keep the code readable and the python interpreter happy!

10

Another example

```
x = 4
y = -3
if x < 0:
    if y > 0:
        print x + y
    else:
        print x - y
else:
    print x * y
```

What does this output?

11

Common if errors

- Syntax errors
 - Mixing up = and == in the condition

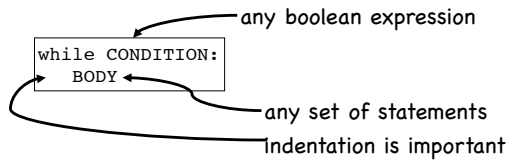
```
b = False
if b = False:
    print b
print 'inside if maybe'
```

SyntaxError: invalid syntax

IndentationError: unindent does not match any outer indentation level

12

The while loop



- As long as the condition is true, the body gets executed repeatedly.
- The first time the condition is false, execution ends.

13

The while loop

```
i = 0
while i < 3:
    print i
    i = i + 1
```

What does this output?

Side note: if the condition is false the first time it is tested, the body is never executed!

14

The break statement

Immediately exits the innermost loop.

```
while True:
    line = raw_input('>>> ')
    if line == 'done':
        break
    print line
print 'Done!'
```

(An if statement is not a loop!)

15

What will happen with this code?

```
i = 0
while i < 3:
    print i
```

It will loop forever (aka Infinite loop)! How do we fix it?

16

The infinite loop

```
i = 4
while i > 0:
    print i
    i = i + 1
```

This code also loops forever!
Why? And how do you fix this?

17

Lists

- A list is a sequence of values.
- Each element (value) is identified by an index.
- The elements of the list can be of any type.

```
tens = [10, 20, 30, 40]
coins = ['dime', 'nickel', 'quarter', 'penny']
empty = []
```

- Lists can have mixed types in them, even other lists (nested).

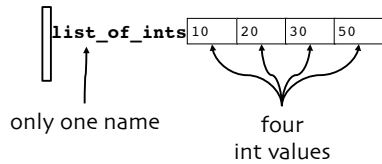
```
mixed = ['hello', 2.0, 5, [10, 20]]
```

18

Creating a list

- Use the `[]` brackets

```
list_of_ints = [10,20,30,50]
```



19

List operators

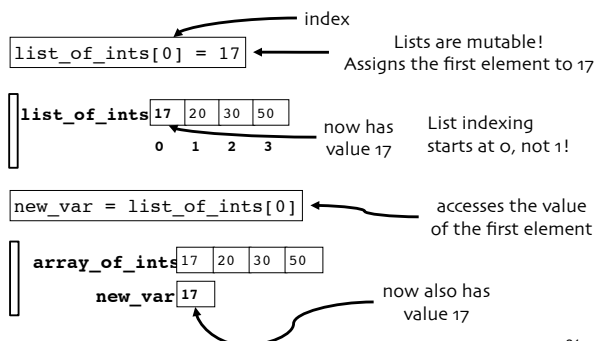
- Applied to lists, produce lists

Operator	Operation
<code>+</code>	Concatenation
<code>*</code>	Repetition
<code><list>[]</code>	Indexing
<code><list>[:]</code>	Slicing

20

Accessing list elements

Individual elements are accessed using the `[]` operator.



21

Printing a list

- We can use the `print` function to output the contents of the list:

```
vocabulary = ['ameliorate', 'castigate', 'defenestrate']
numbers = [17, 123]
empty = []
print vocabulary, numbers, empty
```

22

An example

```
1 numbers = [10, 20, 30]
2 letters = ['a', 'b', 'c']
3 number = numbers[0]
4 letter = letters[2]
5 print 'number =', number
6 print 'letter:', letter
7 print 'letters:', letters
```

23

Another example

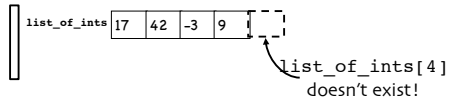
```
1 numbers = [10, 20, 30]
2 letters = ['a', 'b', 'c']
3 mixed = letters + numbers
4 print mixed
5 print letters*2
6 numbers[2] = letters
7 print numbers
8 print numbers[:2]
9 numbers[1:] = [40, 50]
10 print numbers
```

24

Out-of-range errors

- You will get a runtime error if you try to access an element that does not exist!

```
list_of_ints = [17, 9, 42, -2]
print list_of_ints[4]
```



IndexError: list index out of range

25

Lists vs. Strings

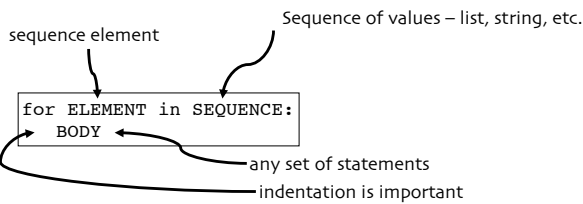
- Lists are mutable - their contents can be modified
- Strings are immutable

```
name = 'Lenny'
name[0] = 'J'
```

TypeError: object doesn't support item assignment

26

The for loop

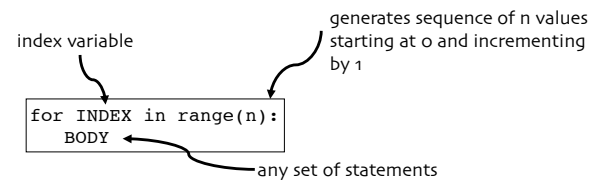


Example:

```
for i in [0,1,2,3]:
    print i
```

27

Using range

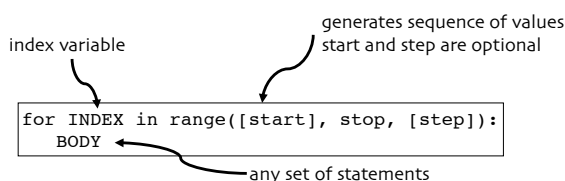


```
for i in range(4):
    sq = i * i
    print i, sq
```

What does this output?

28

Using range



```
for i in range(1, 7, 2):
    print i
```

What does this output?

29

For loop and strings

- Iterating through the characters of a string

```
str1 = 'stressed'
for c in str1:
    print c,
```

- Example

```
str1 = 'stressed'
res = ''
for c in str1:
    res = c + res
print res
```

30

For loop and lists

- Iterating through the elements of a list

```
desserts = [['tiramisu', 3.5], ['baklava', 2], ['creme brulee', 3]]
for [dessert, cost] in desserts:
    print dessert, 'costs $', cost
```

31

Combining for and if

```
for i in range(6):
    if i % 2 == 0:
        print i, 'is even.'
    else:
        print i, 'is odd.'
```

What does this output?

32

Nested for loops

```
for i in range(1,6):
    for j in range(1,6):
        prod = i * j
        # use comma to print all on one line
        print prod,
    print
```

must use
new index
variable for
inner loop

What does this output?

33

For vs While

- For loop is primarily used
 - for iterating over a sequence of values
 - when we know the number of iterations in advance
- While loop is primarily used
 - when we don't know the number of iterations in advance (they could be controlled by user input)

34

Common loop errors

- Semantic errors
 - Forgetting to increment/decrement index variable
 - Incrementing/decrementing in the wrong direction
 - Nested loops: using the same index variable twice!

35

Today's exercises

- Understanding and writing if / else statements
- Understanding and writing while and for loops
- Using lists and for loops
- Practice with debugging loops
- Readings for tomorrow: Sections 3, 6.1-4

36