Fundamentals of Computer Science 30398

Lecture 3

Code blocks

In Python code is blocked by indentation. Watch for errors due to mismatched indentation.

```
n = 15
i = 0
while i < n:
    if i%2 == 0:
        print(i)
    i = i + 1
```

Code blocks

In Python code is blocked by indentation. Watch for errors due to mismatched indentation.

```
n = 15
i = 0
while i < n:
    if i%2 == 0:
        print(i)
    i = i + 1
```

Variables scope

If a variable is first used inside a block, it is local to this block. It will not be visible outside of the block

Example (not working)

```
n = 15
if n%2 == 0:
    is_even = True
print(is_even)
```

Variables scope

If a variable is first used inside a block, it is local to this block. It will not be visible outside of the block

Example (works now)

```
n = 15
is_even = False
if n%2 == 0:
    is_even = True
print(is_even)
```

Boolean operations

Operation	Example	Value
and	$(3 == 5)$ **and** $(3 < 5)$	False
or	$(3 == 5)$ **or** $(3 < 5)$	True
not	**not** $(3 == 5)$	True

Boolean operations

Operation	Example	Value
and	$(3 == 5)$ **and** $(3 < 5)$	False
or	$(3 == 5)$ **or** $(3 < 5)$	True
not	**not** $(3 == 5)$	True

Example

```
if ((x < 10) or (x > 20)) and (x % 2 == 0):
    print(x)
```

Simple for loop (over range)

The following type of loop is very common

```
n = 15
i = 0
while i < n:
    do_stuff
    i = i + 1
```

There is a more idiomatic way of doing this in Python:

```
n = 15
for i in range(n):
    do_stuff
```

Two more things about loops: break, continue

continue instruction jumps to the next iteration of the outermost loop

```
n = 15
for i in range(n):
    if i%2 == 1:
        continue
    print(i)
```

Two more things about loops: break, continue

Beware! This is very different than

```
n = 15
while i<n:
    if i%2 == 1:
        continue
    print(i)
    i = i + 1
```

Two more things about loops: break, continue

break instruction jumps out of the outermost loop

```
for i in range(n):
    do_stuff
    if condition:
        break
    do_stuff_2
```

Implicit truth values

Condition in the if and while statement do not need to be of type **bool**. Some Python values are truthy, some are falsy.

example

```
a = ""
if a:
    print("empty-string-is-falsy")
```

Implicit truth values

Condition in the if and while statement do not need to be of type **bool**. Some Python values are truthy, some are falsy.

example

```
a = ""
if a:
    print("empty-string-is-falsy")
```

How do I best use this

Don't.

Live coding (Exercises 1-5)

Lists

Example

```
a = [1,2,5,"Hello-World!"]
print(a[2])
a[6] = [1,2,3]
print(a)
```

Lists are mutable!

Two variables can reference the same list!

```
a = [1,2,5]
b = a
b[2] = 1
print(a)
```

Three more useful methods of lists

len(list) outputs number of elements in a

```
a = [1,2,5]
print(len(a))
```

a.append() adds an element to the end of the list

```
a = [1,2,5]
a.append(10)
print(a)
```

Three more useful methods of lists

len(list) outputs number of elements in a

```
a = [1,2,5]
print(len(a))
```

a.append() adds an element to the end of the list

```
a = [1,2,5]
a.append(10)
print(a)
```

List with 1000 elements

```
a = [False] * 1000
print(a)
```

For over list

We already know how to iterate over list

```
a = [1,2,5]
for i in range(len(a)):
    print(a[i])
```

For over list

We already know how to iterate over list

```
a = [1,2,5]
for i in range(len(a)):
    print(a[i])
```

New for construction

```
a = [1,2,5]
for x in a:
    print(x)
```

Exercises 6-12