

---

# CS 111: Program Design I

## Lecture 9: Nesting if's, while, open access

---

Robert H. Sloan & Richard Warner  
University of Illinois at Chicago  
Sept. 26, 2019

---

**IS IT TRUE: BOOLEAN  
(PARTLY REVIEW)**

# Recall: Relational Operators

- Give back Boolean (True or False)

== equals

!= not equals

in True if and only if lhs element of rhs

> greater than

>= greater than or equal to

< less than

<= less than or equal to

# Sequence (String & List) Boolean Operators

2 symmetric arguments that apply to all types including sequences:

`==` equals

`!=` is not equal to

**2 arguments; order matters, only for sequences:**

`in` tells if lhs is substring of rhs

e.g., `"c" in "cat" → true`

Unary operator:

`not` negates (flips True/False)

---

# Booleans are an unusual type

- Because there are only two objects of that type!
  - True, False

# Operators on Booleans themselves

- **not**: unary operator (one argument), flips value
  - not True  $\rightarrow$  False; not False  $\rightarrow$  True
- **and**: binary operator (two arguments), True if and only if both its arguments are True
- **or**: binary operator, True if  $\geq 1$  of its arguments is True
  - E.g., True or True  $\rightarrow$  True; True or False  $\rightarrow$  True; False or False  $\rightarrow$  False

# Any questions on project?

- `vig("ABCDEF", "XYZ")`
- `cencrypt(A, X)`
- `cencrypt(B, Y)`
- `cencrypt(C, Z)`
- `cencrypt(D, X)`
- ....
- where  $X = 23$ ,  $Y=24$ ,  $Z=25$ . XYZABC  
(check!)

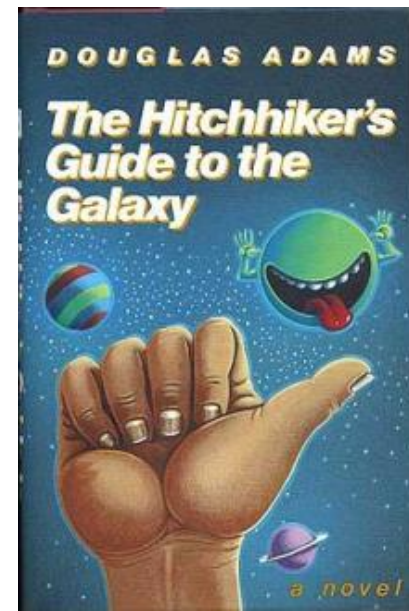
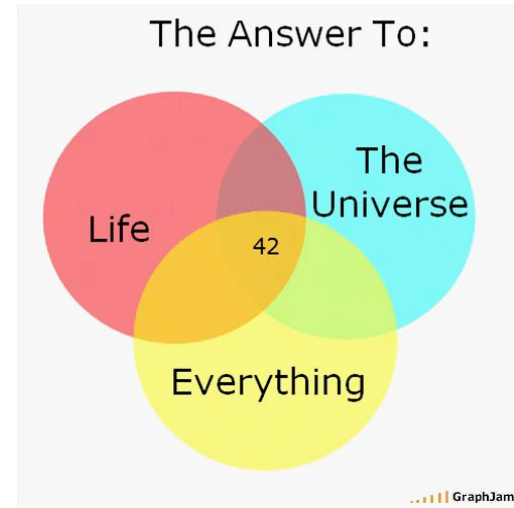


**MORE CONTROL  
STRUCTURES: NESTING IF'S,  
ELIF**



# 3 cases: if, elif, else...

```
def nice(x):  
    """This function demonstrates the use  
    of if, elif, and else"""  
    if x < 42:  
        return "Silly little number!"  
    elif x == 42:  
        return "The answer to life, universe and everything!"  
    else:  
        return "Wow, that's big!"
```



# if, elif, else...

```
def nice(x):  
    """This function demonstrates the use  
    of if, elif, and else"""  
    if x < 42:  
        if x % 2 == 0:  
            return "Silly small even number"  
        else:  
            return "Silly small odd number"  
    elif x == 42:  
        return "Life, universe, everything"  
    else:  
        if x % 2 == 0:  
            return "Big & even"  
        else:  
            return "Big & odd"
```

# if, elif, else..

```
def nice(x):  
    """This function demonstrates the use  
    of if, elif, and else"""  
    if x < 42:  
        if x % 2 == 0:  
            return "Silly small even number"  
        else:  
            return "Silly small odd number"  
    elif x == 42:  
        return "Life, universe, everything"  
    else:  
        if x % 2 == 0:  
            return "Big & even"  
        else:  
            return "Big & odd"
```

What will **nice(42)**  
return?

- A. Silly small even
- B. Silly small odd
- C. Life, universe,  
everything
- D. Big even
- E. Big odd

# if, elif, else..

```
def nice(x):  
    """This function demonstrates the use  
    of if, elif, and else"""  
    if x < 42:  
        if x % 2 == 0:  
            return "Silly small even number"  
        else:  
            return "Silly small odd number"  
    elif x == 42:  
        return "Life, universe, everything"  
    else:  
        if x % 2 == 0:  
            return "Big & even"  
        else:  
            return "Big & odd"
```

What will **nice(43)** return?

- A. Silly small even
- B. Silly small odd
- C. Life, universe, everything
- D. Big even
- E. Big odd

# Classifying things by cases

```
def cfaa(action):  
    """Actually only a few of the MANY cases!"""  
    if no_authorz(action) or exceeds_authorz(action):  
        if relevant_national_security(action):  
            return "1030(a)(1)"  
        elif obtain_info(action):  
            return "1030(a)(2)"  
        elif nonpublic_govt_computer(action):  
            return "1030(a)(3)"  
    elif no_authorz(action):  
        if intend_defraud(action):  
            return "1030(a)(4)"
```



**OPEN ACCESS**

---

# The Need for Better Understanding

- “For law enforcement and prosecutors [and *anyone* concerned with understanding this world and making it better] to do their jobs effectively, many more investigators will need to learn the nitty-gritty details of how contemporary communication technologies work.”
  - Susan Landau, *Listening In: Cybersecurity in an Insecure Age*

---

# Open Access

- “Open access” typically refers to online items that are free of restrictions on access and certain restrictions on use—like copyrights, trademarks, and patents.
- Digital technologies can increase open access to information.
- Web crawlers are a good example.



---

# Web Crawler's Getting Data

- `import urllib.request as ur`
- `connection= ur.urlopen("https://www.cs.uic.edu")`
  - `#Accesses the website`
- `content = connection.read()`
  - `#Copies the data into the computer's memory.`

---

# A Common Pattern

- The pattern begins with information distributed over the Internet.
- *Facebook v. Power Ventures*: the distributed information is social networking sites.
- There are a lot: Facebook, QQ, WeChat, QZone, Tumblr, Instagram, Twitter, Baidu Tieba, Skype, Viber, Sina Weibo and so on.
- Suppose you have accounts on several of those.

---

# More of the Pattern

- You would like a way to post on all of them from just one site.
- Solving that problem was the idea behind Power Venture.
  - You could post on all your sites at once.
- Power Ventures did what many do:
  - collect scattered information
  - make it conveniently accessible.

---

# The Final Part of the Pattern

- The final part: the person or entity in possession of the data objects to the access and claims it is illegal.
- That is what Facebook did.
- The court agreed with them and held that Power Ventures access was illegal.
- Do you think Power Ventures should be able to access Facebook?

# Computer Fraud and Abuse Act

- The CFAA
  - prohibits acts of computer trespass
  - by those who are not authorized users.
- Criminal and civil liability for whoever **(a)** “intentionally accesses a computer **(b)** without authorization . . . , and **(c)** thereby obtains ... information from any . . . computer.”
  - 18 U.S.C. § 1030(a)(2)(C).”

# Unauthorized?

- Power Ventures intentionally accessed Facebook's computers, and
- It obtained information by doing so.
- So: was the access unauthorized?
- Power Ventures did have permission from Facebook's users to access their accounts.
- The court still answers, "not authorized."

# The Courts *Analogy*

- Suppose that a person wants to borrow a friend's jewelry that is held in a safe deposit box at a bank.
- The friend gives permission for the person to access the safe deposit box and lends him a key.
- The person decides to visit the bank while carrying a shotgun.
- The bank ejects the person from its premises and bans his reentry. The gun-toting jewelry borrower could not then reenter the bank, claiming that access to the safe deposit box gave him authority to stride about the bank's property while armed.

# Is That The Right Analogy For This?

- `import urllib.request as ur`
- `connection= ur.urlopen("https://www.cs.uic.edu")`
  - `#Accesses the website`
- `content = connection.read()`
  - `#Copies the data into the computer's memory.`



# Who Makes Money from the Data?

- When you sign up for Connect, you agree to these rules (<https://developers.facebook.com/policy>):
  - Don't sell, license, or purchase any data obtained from us or our services.
  - Don't transfer any data that you receive from us (including anonymous, aggregate, or derived data) to any ad network, data broker or other advertising or monetization-related service.
- So how Power Ventures make enough money?

# hiQ's Business

- hiQ is a data analytics business. It analyzes publicly available data to categorize employees for their employers. Two categories:
  - Keeper—estimates the risk an employee will leave for another job.
  - Skill mapper—summarizes employees' skills
- hiQ *only* uses data from LinkedIn.
  - Without that data, hiQ goes out of business.

# hiQ v. LinkedIn

- hiQ is a data analytics business. It analyzes publicly available data to categorize employees for their employers. Two categories:
  - Keeper—estimates the risk an employee will leave for another job.
  - Skill mapper—summarizes employees' skills
- hiQ *only* uses data from LinkedIn.
  - Without that data, hiQ goes out of business.

---

# The Revocation Letter

- “On May 23, 2017, LinkedIn sent a letter demanding that hiQ “immediately cease and desist unauthorized data scraping and other violations of LinkedIn’s User Agreement.”
  - hiQ was founded in 2012, and had searched LinkedIn since its founding.
  - LinkedIn was acquired by MicroSoft in 2016.

# LinkedIn's Similar Business

- LinkedIn Recruiter

- <https://business.linkedin.com/talent-solutions/recruiter>

- “when they update their profile or celebrate a work anniversary, you’ll receive an update on your homepage. And don't worry - they don’t know you're following them.”

- <https://business.linkedin.com/talent-solutions/blog/2015/04/linkedin-recruiter-tip-use-update-me-to-know-when-to-reach-out-to-prospects>

---

# The CFAA Issue

- “The key question regarding the applicability of the CFAA in this case is whether, by continuing to access public LinkedIn profiles after LinkedIn has explicitly revoked permission to do so, hiQ has “accesse[d] a computer without authorization” within the meaning of the CFAA.”

---

# The Appeals Court Decision

- This month, September 9, the Appeals Court gave its decision on hiQ v. LinkedIn, agreeing with the trial court.



**ITERATION, ITERATION,  
ITERATION. ...**



# Iteration (Iterative code elements)

- Repeat

- for: do once for each letter in a string
  - And other generalizations we will see later
- **while loops**
  - repeat until some condition is met
- Lots of uses. One coming up for us: in web crawler, we will need to say, among other things, "while there are still some URLs left in the text making up this webpage, find me the next on."

# While Loops

```
while <condition>:  
    <body>
```

- While condition is True, execute body statements. Once it is False, continue to next section of code
- Condition must evaluate to either True or False
- Colon and indentation are required

# Example

```
x = 3
```

```
while x < 6:
```

```
    print (x)
```

```
    x = x + 1
```

- x starts as 3; while condition true; 3 is printed
- x becomes 4
- while condition still true
- 4 printed
- x becomes 5
- while condition still true
- 5 is printed
- x becomes 6
- while condition false
- 6 not printed

# This will print out?

```
x = 8
while x > 1:
    print(x)
    x = x // 2
```

- A. 8, 4, 2, 1
- B. 8, 4, 2
- C. 4, 2, 1
- D. 4, 2

# Writing While Loops

```
<initialization>
```

```
while <condition>:
```

```
    <body>
```

```
    <increment or advance>
```

- **Initialize:** Set up variable(s)
- **Condition:** How long should loop run?
- **Body:** What loop does
- **Increment:** Change variable(s) (make sure loop will eventually stop!)
  - Must be *inside* block

---

# General pattern: indentation

- Python control flow constructs
  - e.g., def, for, while, if, else
- Start on own line ending with colon :
- Lines governed by them are indented
- Sometimes called "blocks" or "body"
- Many other languages use { }'s to delineate blocks where Python uses indents

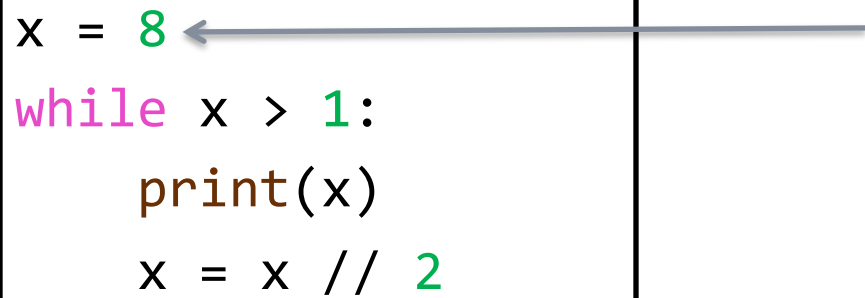
# Example

```
x = 8
while x > 1:
    print(x)
    x = x // 2
```

# Example

initialize

```
x = 8 ←  
while x > 1:  
    print(x)  
    x = x // 2
```





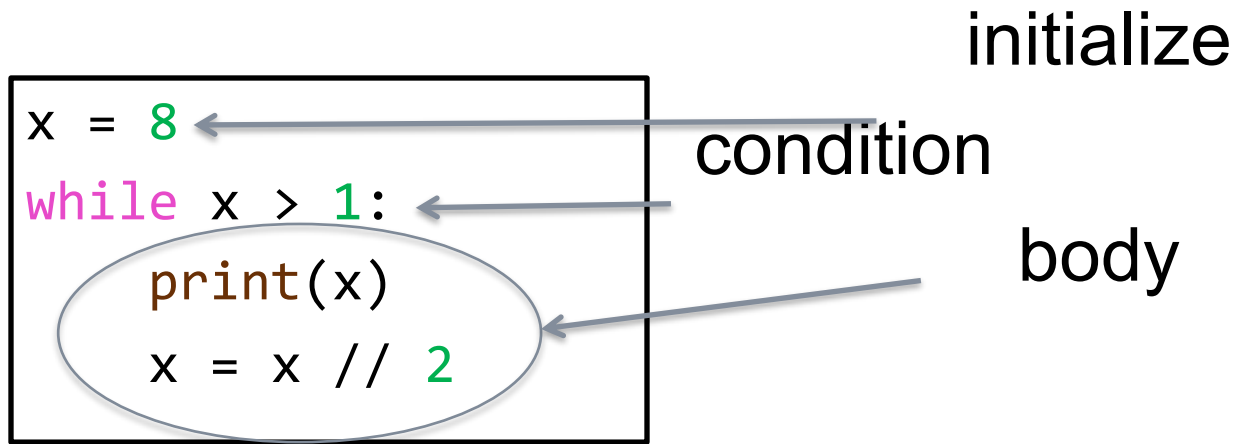
# Example

```
x = 8  
while x > 1:  
    print(x)  
    x = x // 2
```

initialize

condition

# Example



# Example

