Selection & Boolean Logic (Ch. 3)

CSC110: INTRO TO COMPUTER PROGRAMMING WITH PYTHON

INSTRUCTOR: BILL BARRY

1

Roadmap

☞ Selection

Iteration

Midterm Exam (T/F)

More on Strings

File I/O, Exceptions

Lists

List Algorithms

Dictionaries & Sets

Final Exam (T)

2

This Week: Selection and Boolean Logic

Flow of Control, Selection, if Statement

Mutual Exclusivity and elif, Logical Operators

Short Circuits, Boolean Variables/Logic, Boolean Zen

Dependence and Independence

3

Flow of Control, Selection, and the `if` statement

4

Flow of Control

When programs are written, we need ways to say the order in which statements are executed
- This is called **flow of control**

By default, statements are executed top to bottom, one after another
- This is called **sequence**; it's all we've used so far (including when writing functions)

But there are times that's not enough; most programs are not that linear

Programming languages provide ways to alter flow; we'll call those features **control structures**

In the next two weeks we will have covered the three major flow of control:
- **Sequence**
- **Selection** (choices)
- **Iteration** (loops)

These will greatly increase the number of real-world scenarios we can address

5

Analogies

You're walking along Alki Beach with your dog.

Your dog stops at every tree and fire hydrant, one after another, in order
- This is an example of **sequence**; the stops are an orderly progression

If you see dog poop on the path, you step around it
- This is an example of **selection**; you choose a different path based on data

You have a destination in mind; you walk until you reach the coffee shop
- This is an example of **iteration**; you keep repeating an action (taking steps) until a specific goal is reached

6

Selection Basics: Act on One Scenario, Do Nothing in Others

Sometimes we need to act in a specific scenario

Analogy: your little sister says, "The neighbor said she would give me $10 if I mow her lawn"

We can express that in *pseudocode* (English organized like code):

```
lawn mowed?
    get $10
```

It's helpful to introduce options using questions; we will discover later that this is key

A business example: if the purchased item is taxable, add sales tax at the current rate

```
input item price
input whether taxable
taxable?
    item price = item price + (item price x sales tax rate)
```

7

Selection Basics: Do One Thing in One Scenario, Do Another Thing in All Others

Sometimes we need to act in a specific scenario, but do something else in all other scenarios

Analogy: Mom says, "If you have homework, do it; otherwise, you can have one hour of screen time"

```
homework to do?
    do homework
otherwise
    have screen time (woohoo!)
```

A business example: if you hold a stock for less than 12 months, any gain is subject to the *short-term* capital gains rate. Stocks held longer are subject to the *long-term* rate

```
input stock gain (dollars)
input number of months held
held less than 12?
    tax owed = stock gain * 0.20
else
    tax owed = stock gain * 0.15
```

8

Selection Basics: Do Different Things in Various Scenarios

Sometimes we need to act differently in various scenarios

Analogy: Dad handles dinner on Tuesdays. If one of the kids has an activity, we get pizza. If we're having movie night, we order Chinese. Otherwise, he makes spaghetti.

```
kid activity?
    order pizza
movie night?
    order Chinese
otherwise?
    make spaghetti
```

Let's talk about Mutual Exclusivity: one, and only one, of these paths will be taken, guaranteed!

See next slide for business example…

9

Selection Basics: Do Different Things in Various Scenarios

Business example: single people making less than $40K pay a long-term capital gains rate of 0%; if they make $40K to $441,450, 15%; over that, 20%

```
input stock gain
input agi (adjusted gross income)
agi below 40000?
    tax owed = 0
agi up to 441450?
    tax owed = stock gain * 0.15
otherwise…
    tax owed = stock gain * 0.20
```

Let's talk about Mutual Exclusivity: one, and only one, of these paths will be taken, guaranteed!

10

Python Selection Control Structures

Much of the power of computer programs involves the ability to make decisions based on data

Example: if an employee works more than 40 hours, they get time-and-a-half overtime pay

Python has the if statement to help us; as with all block-oriented structures (like functions) it uses **indentation** to indicate what the if covers:

```
pay = hours * wage
if hours > 40:
    pay += (hours – 40) * wage * 0.5
```

Could also be written with an else clause as:

```
if hours <= 40:
    pay = hours * wage
else:
    pay = (wage * 40) + ((hours – 40) * wage * 1.5)
```

11

Boolean Expressions & Relational Operators

Boolean expressions are those whose result is either True or False
- In the previous example: `hours > 40`

To compare values, you can use any of the relational operators:

Operator	Description
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Equal to (note the two equals signs)
!=	Not equal to

This list is common across modern programming languages

A common error is to use one equal sign when you meant two; Python catches that; other languages might not. There's a trick…

12

English to Code: Learn to Interpret!

English	Program
Scores at or above 96% get a grade of A	
If you score 65% or below, you fail the class	
If you spend more than $100, you get a 5% discount	
Those earning at least $100,000 qualify	
Sales up to $100,000 earn 5% commission	

13

English to Code: Learn to Interpret!

English	Program

Scores at or above 96% get a grade of A	`if score >= 0.96:`
	`    letterGrade = 'A'`

If you score 65% or below, you fail the class	`if score <= 0.65:`
	`    letterGrade = 'F'`

If you spend more than $100, you get a 5% discount	`if total > 100:`
	`    discount = total * 0.05`

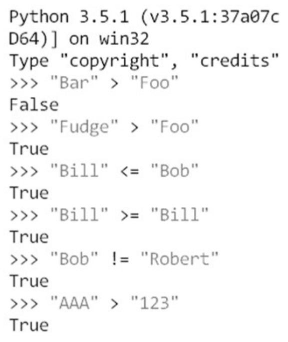
Those earning at least $100,000 qualify	`if earnings >= 100000:`
	`    qualified = 'yes'`

Sales up to $100,000 earn 5% commission	`if sales <= 100000:`
	`    commission = sales * 0.05`

14

Relational Operators with Strings?

What do relational operators do with strings? Can you guess from this Shell session?



```
Python 3.5.1 (v3.5.1:37a07c
D64)] on win32
Type "copyright", "credits"
>>> "Bar" > "Foo"
False
>>> "Fudge" > "Foo"
True
>>> "Bill" <= "Bob"
True
>>> "Bill" >= "Bill"
True
>>> "Bob" != "Robert"
True
>>> "AAA" > "123"
True
```

Think of string comparisons as "dictionary" compares; if string #1 is *later* in the dictionary than string #2, then string #1 is *greater than* string #2. If the strings are the *same*, they are *equal*

Internally, these are done based on numbers that are used to represent characters, ASCII for example:

```
    'A' = 65, 'a' = 97,
    '0' = 48, ' ' = 32
```

Strings are *not* compared by their lengths; don't make that common mistake

15

Pause & Play

LET'S USE SELECTION TO SOLVE A PROBLEM THAT'S BEEN BUGGING US FOR A WHILE: OUT-OF-RANGE USER INPUT

16

Using Selection to Fix Bad User Input

In our recent Turtle code, we asked for pen and pane sizes. What if we want to guard against bad values, e.g., pen size must be at least 1, and pane size must be 10 or more?

OLD CODE

```
def getOptions():
    windowColor = \
        input('What color window? ')
    penSize = \
        int(input('What pen size (1+)? '))
    paneSize = \
        int(input('What size panes (10+)? '))
    return windowColor, penSize, paneSize
```

NEW CODE

```
def getOptions():
    windowColor = input('What color window? ')
    penSize = int(input('What pen size (1+)? '))
    if penSize < 1:
        print("Too small a pen; I'll use 1 instead")
        penSize = 1
    paneSize = int(input('What size panes (10+)? '))
    if paneSize < 10:
        print("Too small a pane; I'll use 10 instead")
        paneSize = 10
    return windowColor, penSize, paneSize
```

17

Mutual Exclusivity and `elif`, Logical Operators

18

Nested Decisions and `elif`

It's not unusual to need more ifs to solve a problem with *mutually exclusive states*, e.g., commission structure is 1% if sales are $1,000 or greater, 5% if sales are $6,000 or greater, and 10% if sales are $10,000 or greater. But the nesting gets hard to read; elif is a better alternative

```
if sales >= 10000:
    commission = sales * 0.10
else:
    if sales >= 6000:
        commission = sales * 0.05
    else:
        if sales >= 1000:
            commission = sales * 0.01
        else:
            commission = 0.0
```

```
if sales >= 10000:
    commission = sales * 0.10
...
```

19

Nested Decisions and `elif`

It's not unusual to need more ifs to solve a problem with *mutually exclusive states*, e.g., commission structure is 1% if sales are $1,000 or greater, 5% if sales are $6,000 or greater, and 10% if sales are $10,000 or greater. But the nesting gets hard to read; elif is a better alternative

```
if sales >= 10000:
    commission = sales * 0.10
else:
    if sales >= 6000:
        commission = sales * 0.05
    else:
        if sales >= 1000:
            commission = sales * 0.01
        else:
            commission = 0.0
```

```
if sales >= 10000:
    commission = sales * 0.10
elif sales >= 6000:
    commission = sales * 0.05
elif sales >= 1000:
    commission = sales * 0.01
else:
    commission = 0.0
```

20

Logical Operators in Real Life

These aren't strange concepts; we use them all the time:

➔ If you clean your room **and** wash your face, you can have dessert

➔ If you take out the trash **or** sweep the porch, we'll go to the game

What do we mean when we use these conjunctions?

Which is more *restrictive*, and which more *permissive*?

We also use *not* to negate an otherwise positive statement:

➔ I said that if you snuck onto the computer one more time, you were **not** going to the party

21

Logical Operators in Code

Combine Boolean expressions into more complex ones by using logical operators and, or, and not. It's common to express results of these operators as Truth Tables:

x	**y**	**x and y**
True	True	True
True	False	False
False	True	False
False	False	False

x	**y**	**x or y**
True	True	True
True	False	True
False	True	True
False	False	False

x	**not x**
True	False
False	True

Examples:

```
if sales >= 15000 and prevMonthSales >= 15000:
if commission < 8000 and onProbation == True:
if (x > y and y % 5 == 0) or x < 0:
```

Important note: you need a full Boolean expression on both the **left and right sides** of a logical operator; a common error is to forget that, e.g.,

```
if x < 0 or > 20  # not right!
```

22

Pause & Play

LET’S USE SELECTION TO SOLVE A PROBLEM THAT’S BEEN BUGGING US FOR A WHILE: OUT-OF-RANGE USER INPUT

23

Fixing More Bad Input: Range Checking

Update getOptions to also handle bad colors (allow RGB, for simplicity), pen sizes too large (over 20), and panes too large (over 300); instead set the values to defaults you choose

```
def getOptions():
    windowColor = input('What color window? ')
    penSize = int(input('What pen size (1+)? '))
    if penSize < 1:
        print("Too small a pen; I'll use 1 instead")
        penSize = 1
    paneSize = int(input('What size panes (10+)? '))
    if paneSize < 10:
        print("Too small a pane; I'll use 10 instead")
        paneSize = 10
    return windowColor, penSize, paneSize
```

24

Range Checking: Updated Code

```
def getOptions():
    windowColor = input('What color would you like the window to be? ')
    if         windowColor != 'red'   \
           and windowColor != 'green' \
           and windowColor != 'blue':
        print("Bad window color; I'll use blue instead")
        windowColor = 'blue'
    penSize = int(input('What pen size would you like (1-20)? '))
    if penSize < 1 or penSize > 20:
        print("Bad pen size; I'll use 1 instead")
        penSize = 1
    paneSize = int(input('What size panes would you like (10-300)? '))
    if paneSize < 10 or paneSize > 300:
        print("Bad pane size; I'll use 100 instead")
        paneSize = 100
    return windowColor, penSize, paneSize
```

What about caps or mixed-case entries?

If time, a quick tangent on De Morgan's law https://en.wikipedia.org/wiki/De_Morgan%27s_laws

25

Pause & Play

LET'S WORK IN GROUPS TO CODE A PROBLEM SOLUTION AND TEST OUR CODE

26

Cascadia College CS Program Criteria

Introduction

Cascadia College's CS program only accepts students who meet these criteria:
- High school equivalency: have a high school diploma; alternatively, have a GED
- Prerequisites: completed CSE 113 with a score of at least 3.5; alternatively, scored 4.0 or higher on the CS AP (advanced placement) test
- Able to start in the fall

Reminder
- When working in groups, *speed* isn't the only goal. Everyone should participate, no one should overpower the group or run the whole show. Follow "we all rise together"

Tasks
- Ask the user five questions (sample code in chat)
  - Bonus if you can be flexible on casing, e.g., Y or y are both accepted as positive answers
- Use the criteria to decide whether the student will be accepted or denied; display the result
- List the test cases necessary to test this code; a table might be a good way to show them

27

Cascadia College: Bill's Solution

```
haveDiploma  =       input("Have diploma (Y/N): ").upper()
haveGed      =       input("Have GED (Y/N): ").upper()
readyForFall =       input("Able to start in Fall 2021 (Y/N): ").upper()
csc113Grade  = float(input("CSC 113 grade (0 if not taken): "))
apTestScore  = float(input("CS AP Test Score (0 if not taken): "))


if          (haveDiploma == "Y" or haveGed == "Y")      \
        and readyForFall == "Y"                         \
        and (csc113Grade >= 3.5 or apTestScore >= 4.0):
    print("Accepted")
else:
    print("Denied")
```

28

Cascadia College: Bill's Tests

Diploma	**GED**	**Fall Start**	**CSC113 Grade**	**AP Test Score?**	**Expected**
Y	**N**	Y	3.5	4.0	Accepted
N	**Y**	Y	3.5	4.0	Accepted
N	**N**	Y	3.5	4.0	Denied
Y	N	**N**	3.5	4.0	Denied
Y	N	Y	**3.4**	**4.0**	Accepted
Y	N	Y	**3.5**	**3.9**	Accepted
Y	N	Y	**3.4**	**3.9**	Denied

29

Short Circuits, Boolean Variables & Logic, Boolean Zen

30

Short-circuit Evaluation

Analogy

On a hike, you say to your friend, "If the ledge is stable and you can get a good picture of the river, please take a picture." In this scenario, if the ledge doesn't seem stable, do you check if a good picture is available? No! Instead, you "short circuit" the request

Short Circuits in Code

In most languages, the binary logical operators will short circuit. With `and`, if the first expression evaluates to `False`, there's no point in checking further. With `or`, if the first expression evaluates to `True`, there's no point in checking further

In certain cases, these are super helpful; the first expression acts as a *guard* to keep subsequent expressions from being checked and getting you into trouble. Example: "if the list isn't empty and the first item is 36, then…" where checking item 1 contents would blow up on an empty list

31

Boolean Variables

A Boolean **variable** is one whose value can be either **`True`** or **`False`**. A Boolean **expression** is one that evaluates to either **`True`** or **`False`** (note the casing of the literals)

Booleans often serves as flags (indicators) of whether some specific state exists

Naming these with a **verb prefix** helps us to remember they are Boolean variables:

```
hasGraduated = True
owesMoney = False
isOnHonorRole = True
isTurtleUp = True
```

This leads to interesting code, like…

```
owesMoney = not owesMoney         # what does this do?
```

32

Pause & Play

LET'S WRITE TOGETHER, USING BOOLEAN VARIABLES IN A SIMPLE PAYROLL PROGRAM

33

Payroll

Ask the user for employee hours and wage

Create a variable that contains **`True`** if the employee worked overtime (more than 40 hours), **`False`** if not

Use that variable to calculate gross pay (before deductions):

- If the employee worked no overtime, then gross pay is hours x wage
- If the employee worked overtime, then gross pay is hours x wage for 40 hours, plus hours x wage x 1.5 for the overtime hours (aka "time and a half")

Print out information like this:

```
Enter employee hours: 35
Enter employee wage per hour: 17.98

The employee worked 35.0 regular hours
Gross pay is $629.30
```

```
Enter employee hours: 55
Enter employee wage per hour: 17.98

The employee worked 15.0 overtime hours and 40 regular hours
Gross pay is $1,123.75
```

34

Payroll: Bill’s Solution

```
empHours = float(input("Enter employee hours: "))
empWage = float(input("Enter employee wage per hour: "))

if empHours > 40:
    overTime = True
else:
    overTime = False

print()
print("The employee worked ", end='')
if overTime == True:
    overtimeHours = empHours - 40
    print(overtimeHours, "overtime hours and 40 regular hours")
    grossPay = (40 * empWage) + (overtimeHours * empWage * 1.5)
else:
    print(empHours, "regular hours")
    grossPay = empHours * empWage

print("Gross pay is $", format(grossPay, ",.2f"), sep='')
```

Note: this isn’t the final/best solution; we need to learn one more thing…

35

Boolean Zen: A State of Simple Elegance for Working with Booleans

If we store Boolean data in variables, the contents are *already* True or False; we don’t need to compare them with Boolean literals. Consider this non-Zen code:

```
isMinor = True
if isMinor == True:
    print("No alcoholic drinks for you!")
else:
    print("Okay to order alcoholic drinks")
```

The IF statement above has a redundancy; it is like saying “if the truth is true.” We can simplify to:

```
if isMinor:          # ah, Boolean Zen!
```

We can also assign Boolean contents to variables using Zen:

```
isMinor = age >= 21  # evaluate Boolean expression, assign result
```

36

Zen or Not So Zen?

```
# This code has not reached Zen

if hours > 40:
      workedOvertime = True
else:
      workedOvertime = False

...


if workedOvertime == True:
```

```
# This code has reached Zen

workedOvertime = hours > 40





if workedOvertime:
```

37

Boolean Parameters & Returns

In functions you can have Boolean parameters and returns

```
def displayJacket(isRaining):
    if isRaining:
        print("Wear waterproof jacket")
    else:
        print("Wear regular jacket")
print(displayJacket(True))            # Wear waterproof jacket
```

```
def acceptNewStudent(gpa):
    return gpa > 3.5
print(acceptNewStudent(3.49))         # False
```

38

Pause & Play

LET’S RETURN TO OUR PAYROLL PROGRAM AND ACHIEVE BOOLEAN ZEN

39

Practice Problems: Code and Test

Write a function called **estimateCusts** to help Pete’s Pasta that returns the number of customers to expect

The function should take Boolean parameters indicating:

- Whether it’s a weekend
- Whether it will be a rainy day
- Whether there is a pro sports game

On a normal weekday, Pete’s expects about 200 customers; on a normal weekend, 350. Rain keeps about 10% of customers away. Games keep about 15% of customers away. These are cumulative, e.g., rainy game days take away 10%, then 15% of remainder

Write a function called **canBirdFly** that takes one parameter, the name of the bird species, and returns a Boolean indicating whether the birds of the specified species can fly

List of common flightless birds:

- Ostrich
- Emu
- Cassowary
- Kiwi
- Rhea
- Penguin

Bonus: be forgiving about parameter casing

40

Practice Problems: Bill’s Solutions

```
def estimateCusts(isRainy, isGame, isWeekend):

    if isWeekend:
        custs = 350
    else:
        custs = 200

    if isRainy:
        custs *= (1 - .1)

    if isGame:
        custs *= (1 - .15)

    return int(custs)
```

```
def canBirdFly(species):
    species = species.lower()
    return not (species == 'ostrich'   \
             or species == 'emu'       \
             or species == 'cassowary' \
             or species == 'kiwi'      \
             or species == 'kiwi'      \
             or species == 'rhea'      \
             or species == 'penguin')
```

41

Dependence and Independence

42

Intro

We're starting to get the hang of using **if** and **elif** to help with mutually exclusive scenarios; these are great for straightforward problems like this one regarding apartment pet deposits

Dogs under 50 pounds are allowed in your apartment. Tenants may have no more than 3 dogs. Pet deposits are as follows:

- 0 dogs → $0.00; 1 dog → $100.00; 2–3 dogs → $75.00 per dog

We can code this using if/elif/else, as shown at right

But what if the scenario gets more complicated, and has more factors to consider? As complexity increases, our brains tend to overcomplicate the scenarios (and therefore, the code). We must recognize the dependence (or lack thereof) in the factors involved

```
if dogCount == 0:
    deposit = 0
elif dogCount == 1:
    deposit = 100
else:
    deposit = 75 * dogCount
```

43

Dependence vs. Independence

We can get into trouble if we confuse **independent** portions of problems with **dependent** ones. Consider a lease clause regarding a non-refundable cleaning deposit:

- For dogs, the deposit is $50 per dog
- For cats, the deposit is $100 per cat

How will we calculate the deposit (being careful not to handle negative numbers)?
If we're not careful we go crazy writing…

```
    if dogCount = 1 and catCount = 1:
        deposit = 50 * 1 + 100 * 1
    elif dogCount = 2 and catCount = 1:
        deposit = 50 * 2 + 100 + 1
    ...
```

…when we're missing the fact that dogs and cats are *independent* calculations; they can be handled in *sequence*, separate from each other

44

Independence: Code Solution

```
def calcDeposit(dogCount, catCount):
    dogDeposit = 0
    catDeposit = 0
    if dogCount > 0:
        dogDeposit = dogCount * 50
    if catCount > 0:
        catDeposit = catCount * 100
    totalDeposit = dogDeposit + catDeposit
    return totalDeposit

def main():
    dogs = int(input('How many dogs? '))
    cats = int(input('How many cats? '))
    dep = calcDeposit(dogs, cats)
    print('The non-refundable cleaning deposit will be $', dep)

main()
```

What are the interesting test cases, here?

# dogs	# cats	expect
-1	0	0
0	-1	0
0	0	0
1	0	50
0	1	50
1	1	150
3	8	950

45

Dependence vs. Independence, Part 2

Now consider this version:
- No more than three pets are allowed per apartment
- If the tenant has only dogs, the deposit is $50 per dog; only cats, $100 per cat
- If the tenant has both dogs and cats, the deposit is $125 per animal

Again, we can convince ourselves a *lot* of complex code is necessary; it's not!

We need to recognize we have four mutually exclusive states: (1) too many pets, (2) cat and dog combo, (3) only dogs, (4) only cats

Let's update our code to handle this scenario…

46

Independence #2: Code Solution

```
def calcDeposit(dogCount, catCount):
    if (dogCount + catCount) > 3:
        # too many pets
        deposit = 9999
    elif dogCount > 0 and catCount > 0:
        # combination of dogs and cats
        deposit = 125 * (dogCount + catCount)
    elif dogCount > 0:
        # dogs only
        deposit = 50 * dogCount
    else:
        #cats only
        deposit = 100 * catCount
    return deposit

def main():
    dogs = int(input('How many dogs? '))
    cats = int(input('How many cats? '))
    dep = calcDeposit(dogs, cats)
    if dep == 9999:
        print('I am sorry; we cannot lease to you')
    else:
        print('The non-refundable cleaning deposit will be $', dep)

main()
```

What are the interesting test cases, here?

# dogs	# cats	expect
4	0	9999
0	4	9999
2	2	9999
2	1	200
1	2	250
3	0	150
0	3	300

47

The End

48