CS 110
If-statements

Benjamin Dicken

Announcements

- Keep up with the mmaterial

Style Guide

The if-statement

- If statements can be used to **run code conditionally**
  - **Before if-statements:** Code has pretty much just run in a straight line
  - **With ifs:** Can run code optionally, depending on the value of a condition
- This means our code can **branch** in different directions

```
if condition:
    statement 1
    statement 2
    . . .
    statement N
```

Determining Boxing weight class

- Write a program that accepts one number (a person's weight in lbs)
- Determines if that person is a **flyweight**, **heavyweight**, or within an **in-between weight class**
- https://en.wikipedia.org/wiki/Weight_class_%28boxing%29

Divisions	Weights
Heavyweight	200+ lbs
Light heavyweight	168–175 lbs
Middleweight	154–160 lbs
Welterweight	140–147 lbs
Lightweight	130–135 lbs
Featherweight	122–126 lbs
Bantamweight	115–118 lbs
Flyweight	108–112 lbs

Determining Boxing weight class

- Write a program that accepts one number (a person's weight in lbs)
- Determines if that person is a **flyweight**, **heavyweight**, or within an **in-between weight class**
- https://en.wikipedia.org/wiki/Weight_class_%28boxing%29

Division	Weight
Heavyweight	200 + lbs
Mediumweight	Between 108 and 200 lbs
Flyweight	108 or less lbs

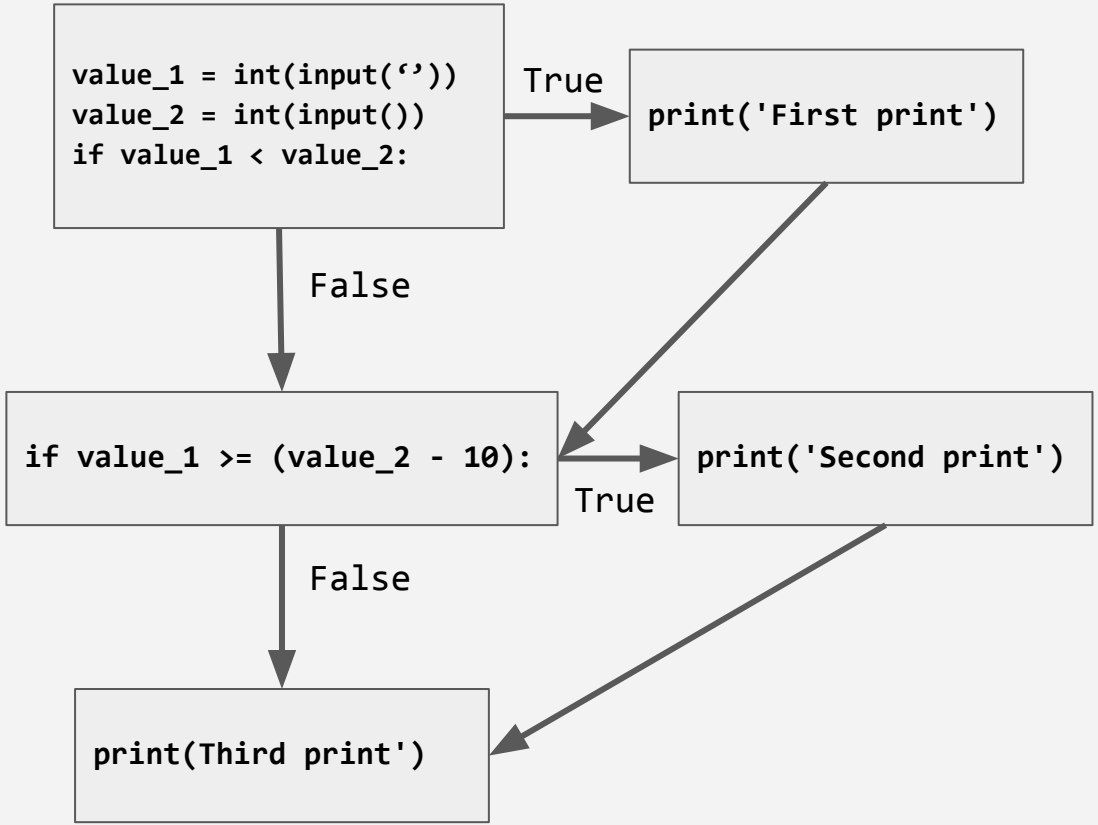
Announcements

- Video Materials
- Log in to Gradescope
- Labor Day

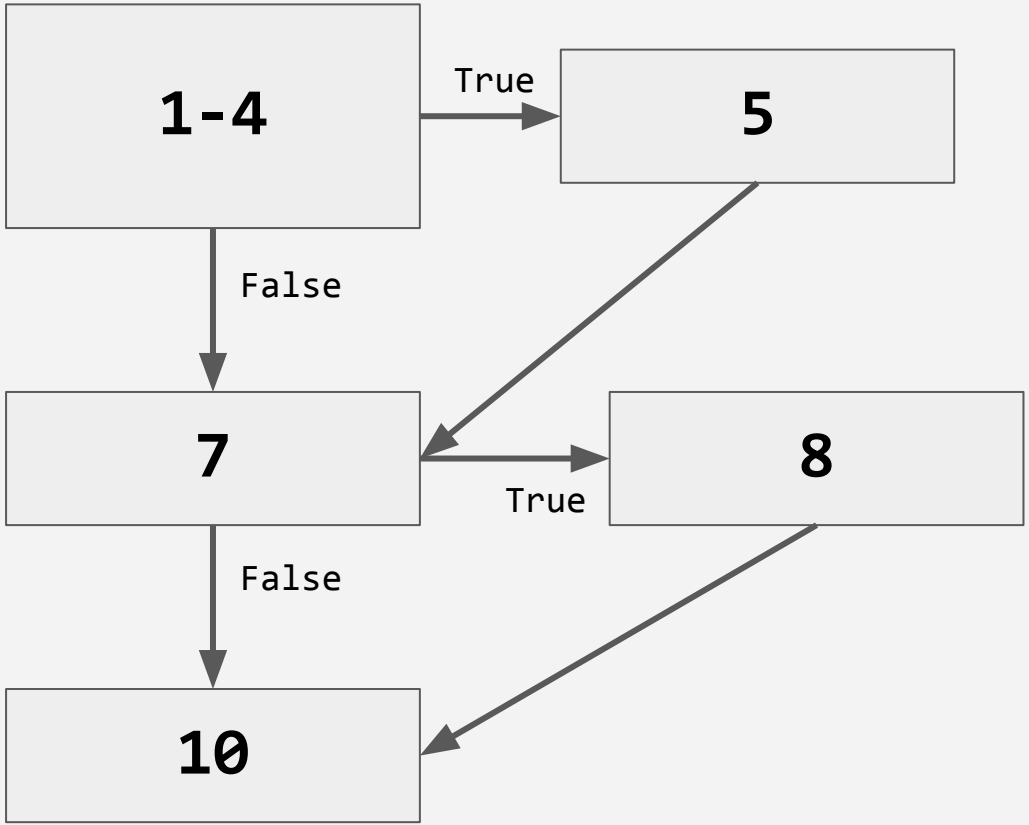
What is a control-flow graph (CFG)

- A diagram that breaks down the code into all chunks that will always run in sequence, and shows the possible paths that can be taken
- Along the lines of decision structure

```
value_1 = int(input(‘’))
value_2 = int(input(‘’))

if value_1 < value_2:
    print('First print')

if value_1 >= (value_2 - 10):
    print('Second print')

print('Third print')
```



```
1  value_1 = int(input(‘’))
2  value_2 = int(input(‘’))
3
4  if value_1 < value_2:
5      print('First print')
6
7  if value_1 >= (value_2 - 10):
8      print('Second print')
9
10 print('Third print')
```



Ifs and Prints

Draw the CFG

```
age = int(input('How old are you? \n'))

if age >= 18:
    print('You may apply to join the military')
if age >= 21:
    print('You may drink')
if age > 35:
    print('You may run for president')
```

Ifs and Prints

What happens when the user types in a non-integer?

```
age = int(input('How old are you? \n'))

if age >= 18:
    print('You may apply to join the military')
if age >= 21:
    print('You may drink')
if age > 35:
    print('You may run for president')
```

Checking for numbers

- You can use the function **isnumeric()** to determine if a string represents a number
- Makes sure a string contains only digits

Checking for numbers

- You can use the function **isnumeric()** to determine if a string represents a number
- Makes sure a string contains only digits
- For example:

```
name = 'Jimmy'
name.isnumeric()
```

```
age = 37
age.isnumeric()
```

Ifs and Prints

What, if any, input values would cause this program to print out no text?

```
par = int(input('Golf hole par: '))
swings = int(input('Swings on hole: '))

if swings >= 7:
    print('Go get golf lessons')
if swings == par:
    print('On Par!')
if swings <= (par-1):
    print('Wow, under par!')
```

Exiting a Program

- Sometime, when you encounter a problem or a bad input value, you want your program to exit, without running any additional code
- How?

Exiting a Program

- Sometime, when you encounter a problem or a bad input value, you want your program to exit, without running any additional code
- How?

```
exit()
```

Stops the program

Exiting a Program

- Sometime, when you encounter a problem or a bad input value, you want your program to exit, without running any additional code
- How?

```
print('antelope')
exit()
print('artichoke')
```

Stops the program

Ifs and Prints

Modify so that if the input values are not numeric, the program will print a message and then exit

```
par = int(input('Golf hole par: '))
swings = int(input('Swings on hole: '))

if swings >= 7:
    print('Go get golf lessons')
if swings == par:
    print('On Par!')
if swings <= (par-1):
    print('Wow, under par!')
```

```
par = input('Golf hole par: ')
if par.isnumeric() != True:
    print('Invalid par')
    exit()
par = int(par)

swings = input('Swings on hole: ')
if swings.isnumeric() != True:
    print('Invalid swing amount')
    exit()
swings = int(swings)
```

```
if swings >= 7:
    print('Go get golf lessons')
if swings == par:
    print('On Par!')
if swings <= (par-1):
    print('Wow, under par!')
```

```
par = input('Golf hole par: ')
if par.isnumeric() != True:
    print('Invalid par')
    exit()
par = int(par)
swings = input('Swings on hole: ')
if swings.isnumeric() != True:
    print('Invalid swing amount')
    exit()
swings = int(swings)

if swings >= 7:
    print('Go get golf lessons')
if swings == par:
    print('On Par!')
if swings <= (par-1):
    print('Wow, under par!')
```

What if we want to add an additional restriction?

Perhaps, the par must less than 7 and greater than 1.

Try it!

```
par = input('Golf hole par: ')
if par.isnumeric() != True:
    print('Invalid par')
    exit()
par = int(par)
if par < 1:
    print('Enter a realistic par.')
    exit()
if par > 6:
    print('Enter a realistic par.')
    exit()

# (also get swing count)

if swings >= 7:
    print('Go get golf lessons')
if swings == par:
    print('On Par!')
if swings <= (par-1):
    print('Wow, under par!')
```

What if we want to add an additional restriction?

Perhaps, the par must less than 7 and greater than 1.

Try it!

Relational Operators

What are they?

Relational Operators

- Can be used to compare two variables or expressions
- Compare the **left-hand side** with the **right-hand side**, and then evaluate to either **True** or **False**

Relational Operators

- Can be used to compare two variables or expressions
- Compare the **left-hand side** with the **right-hand side**, and then evaluate to either **True** or **False**

`==` `!=` `>=` `>` `<=` `<`

Combining conditions with **and**

- You can place the **and** keyword in-between two expressions that evaluate to a boolean (True or False)
- The **and** will combine the two sides, and will result in **True** only if both sides are also **True**.
  - Otherwise **False**

one **and** two

	one: True	one: False
two: True	True	False
two: False	False	False

What would it print?

```
a = (4 > 4) and (3 <= 7)
b = a and ((5 - 12) != 4)
print(a and b)
```

Combining conditions with **or**

- You can place the **or** keyword in-between two expressions that evaluate to a boolean (True or False)
- The **or** will combine the two sides, and will result in **False** only if both sides are also **False**.
  - Otherwise **True**

one **or** two

two \ one	True	False
True	True	True
False	True	False

What would it print?

```
a = (4 == 4) or (3 > 7)
b = a and False or True
print(a or b)
```

Duplication

Notice the duplication

Can this code be made more compact?

```
if par < 1:
    print('Enter a realistic par.')
    exit()
if par > 6:
    print('Enter a realistic par.')
    exit()
```

Activity

```
par = input('Golf hole par: ')
if par.isnumeric() != True:
    print('Invalid par')
    exit()
par = int(par)
# . . .
if par < 1:
    print('Enter a realistic par.')
    exit()
if par > 6:
    print('Enter a realistic par.')
    exit()

if swings >= 7:
    print('Go get golf lessons')
if swings == par:
    print('On Par!')
if swings <= (par-1):
    print('Wow, under par!')
```

How could this code be made more compact using the and / or boolean operators?

Activity

```
par = input('Golf hole par: ')
if par.isnumeric() != True:
    print('Invalid par')
    exit()
par = int(par)
# . . .
if par > 6 or par < 1:
    print('Enter a realistic par.')
    exit()

if swings >= 7:
    print('Go get golf lessons')
if swings == par:
    print('On Par!')
if swings <= (par-1):
    print('Wow, under par!')
```