

ITP 115 – Programming in Python

Branching and Modules

Consider Instagram Login

Instagram

Username

Password

[Forgot?](#)

Log in

Correct Account Info

Email: **ttrojan@usc.edu**


Password: **traveler**

1. How do we check if the password is correct?
2. If the password is correct, what should we do?
What if it is incorrect?

Program Flow



```
correctUserName = "ttrojan"  
correctPassword = "traveler"
```



```
username = input("Please enter your name: ")  
password = input("Please enter your password: ")
```




```
print("Welcome", username)
```

Program Flow



```
password = input("Please enter your password: ")
```

```
print("Welcome", username)
```

- 
- But what if the password is incorrect?
 - We have no way to make *decisions* or change the *flow of control*

Flow of control

- The order a program performs actions
- Up to now our programs have been sequential
- **Branching statements** choose between 2 or more possible actions

Branching

- Fundamental part of computer programming
- Making a decision to take one path or another
- Use the **if** structure
- All **if** structures have a **condition**
 - Think of a **condition** like a "Yes or No" Question

The condition

if number > 1:

- Conditions evaluate to **True** or **False**
 - An expression that evaluates to **True** or **False** is a **boolean expression**
 - Operators that evaluate to **True** or **False** are called **boolean operators**

Syntax

- Place a colon **:** after the **condition:**
- Indent the lines underneath the **if** statement
- There is also an optional **else**

```
if condition:  
    statement1
```

```
if condition:  
    statement1  
else:  
    statement2
```


Examples

```
password = input("Enter your password: ")  
if password == "secret":  
    print("Access Granted")  
else:  
    print("Access Denied")
```

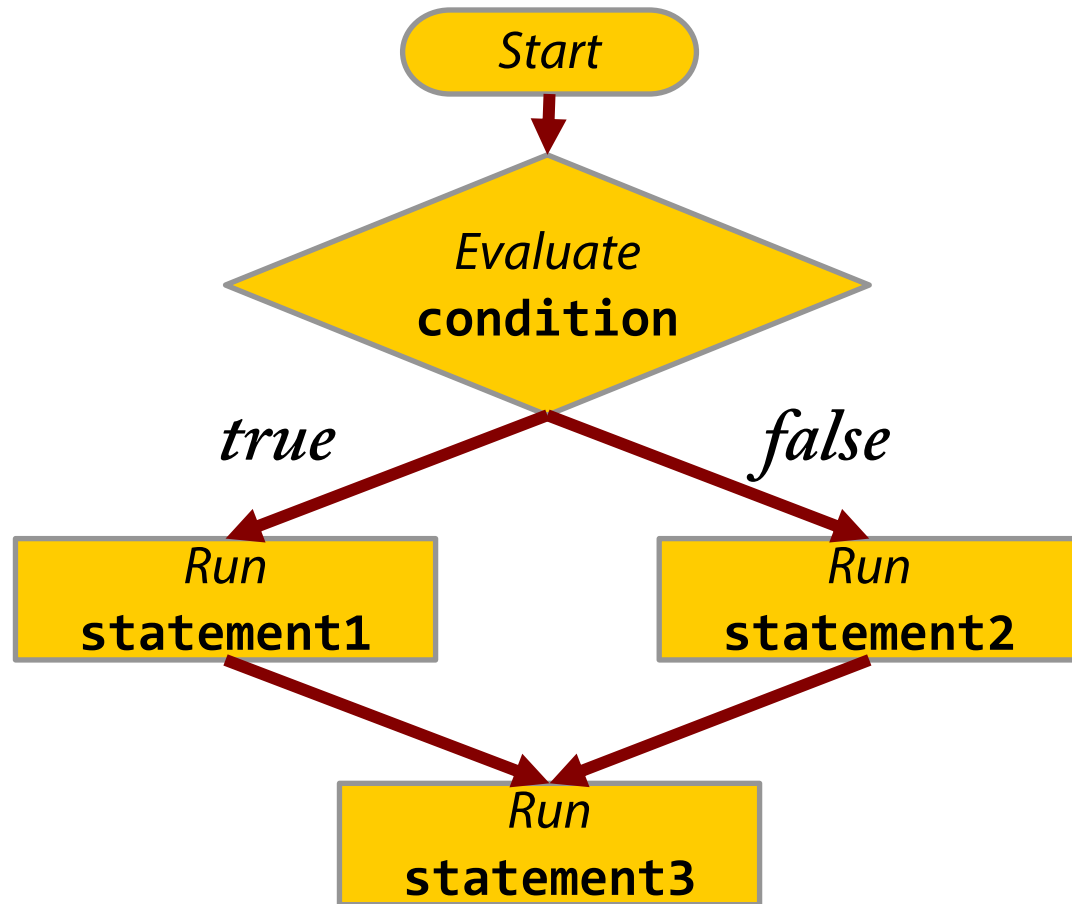
```
age = int(input("Enter your age: "))  
if age >= 18:  
    print("You can vote!")  
else:  
    print("Not yet")
```

Semantics of **if** - **else**

if condition:
 statement1

else:
 statement2

statement3



Semantics of **if** - **else**

if **condition**:
 statement1

else:
 statement2

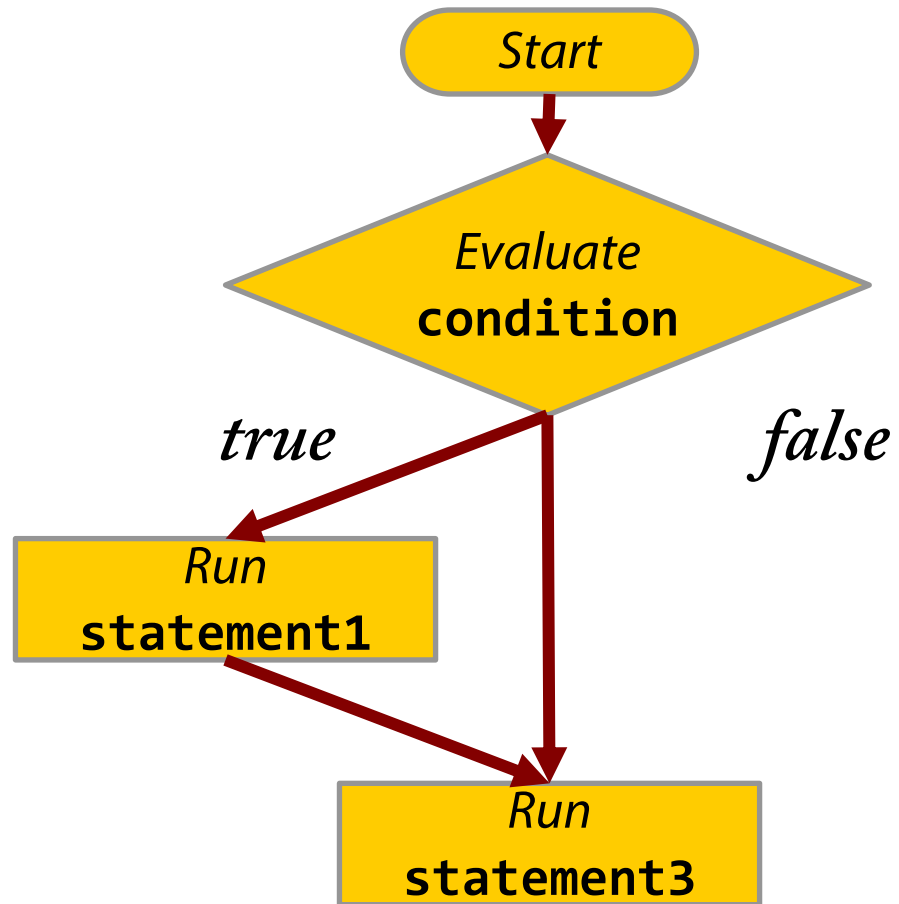
statement3

- **statement1** happens if **condition** is **true**
- **statement2** happens if **condition** is **false**
- **statement3** is always executed afterwards
- **statement1** and **statement2** will never execute together

Skipping else

**if condition:
statement1**

statement3



Question

- What if we want more than one statement to execute if a condition is true?

Create Blocks

- A **block** is one or more consecutive lines indented by the same amount
- Indenting sets lines off not only visually, but logically too
 - Together they form a single unit
- Indenting to create blocks is **not optional**
 - It's the only way to define a block

Block Example

```
favFood = input("Enter favorite food: ")
if favFood == "pizza":
    print("Your favorite food is pizza")
    favTopping = input("Enter favorite topping")
    if favTopping == "sausage":
        print("Your favorite topping is sausage")
        print("Me too")
    else:
        print("Your favorite topping is", favTopping)
print("Have a great day!")
```

Block Example

```
favFood = input("Enter favorite food: ")
```

```
if favFood == "pizza":
```

```
    print("Your favorite food is pizza")
```

```
    favTopping = input("Enter favorite topping")
```

```
        if favTopping == "sausage":
```

```
            print("Your favorite topping is sausage")
```

```
            print("Me too")
```

```
        else:
```

```
            print("Your favorite topping is", favTopping)
```

```
print("Have a great day!")
```

Every indented line in a block is grouped

How is this different?

```
favFood = input("Enter favorite food: ")
if favFood == "pizza":
    print("Your favorite food is pizza")
    favTopping = input("Enter favorite topping")
    if favTopping == "sausage":
        print("Your favorite topping is sausage")
        print("Me too")
else:
    print("Your favorite topping is", favTopping)
print("Have a great day!")
```

Comparison Operators

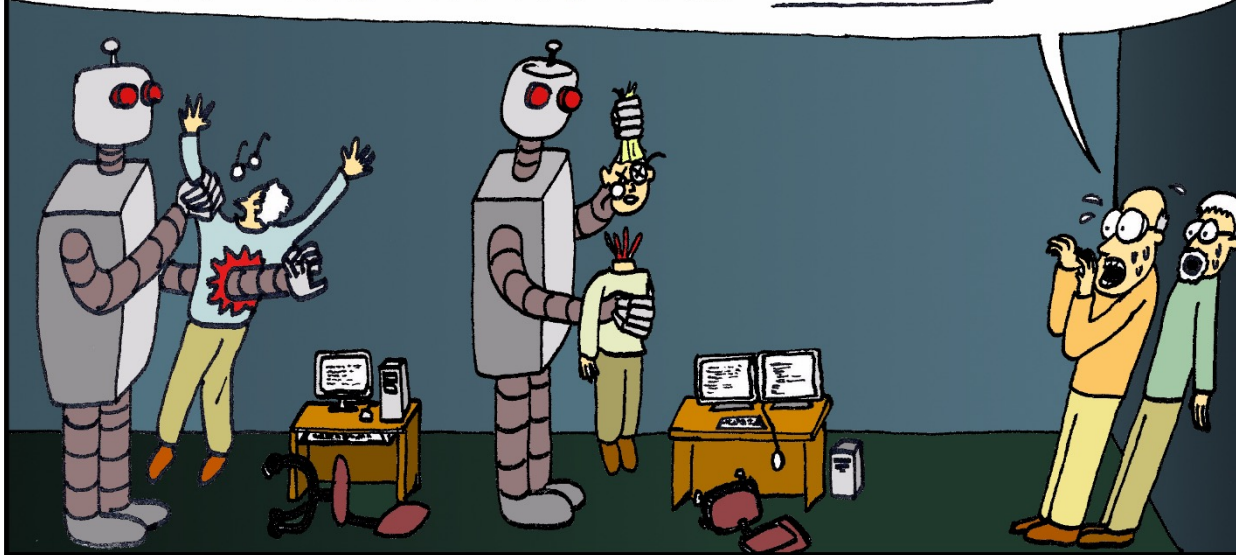
Operator	Meaning	Sample Condition	Evaluates To
<code>==</code>	equal to	<code>5 == 5</code>	True
<code>!=</code>	not equal to	<code>8 != 5</code>	True
<code>></code>	greater than	<code>3 > 10</code>	False
<code><</code>	less than	<code>5 < 8</code>	True
<code>>=</code>	greater than or equal to	<code>5 >= 10</code>	False
<code><=</code>	less than or equal to	<code>5 <= 5</code>	True

If you compare strings, you get results based on alphabetical order

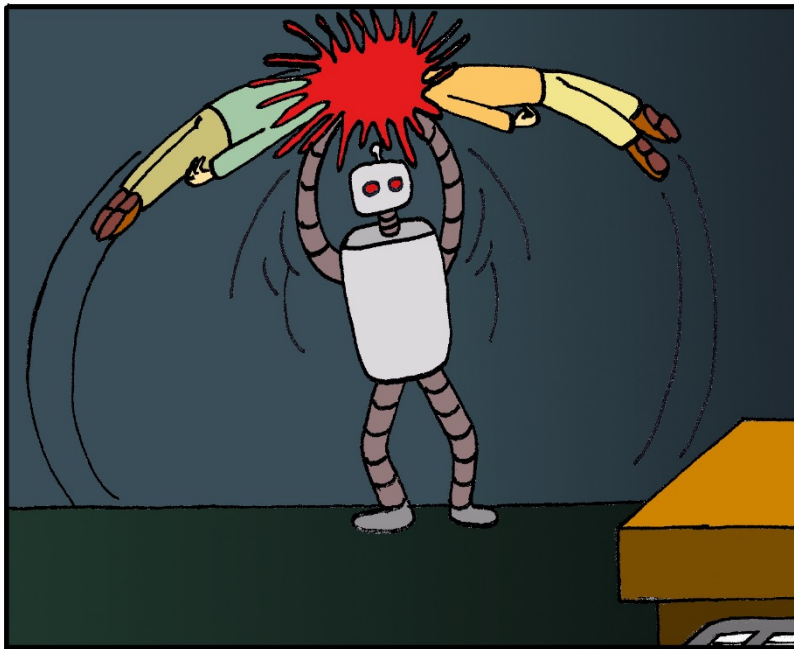
- *End of session 1*

Review

OH NO! THE ROBOTS ARE KILLING US!!!



BUT WHY?!!? WE NEVER PROGRAMMED THEM TO DO THIS!!!



```
isMurderingRobot = False
```

```
def interactWithHumans():  
    if isMurderingRobot = True:  
        kill(humans)  
    else:  
        beNiceTo(Humans)
```

Question

- What if we want to choose between more than 2 options?

Multibranch **if-elif** Statements

```
if condition1:
    statement1
elif condition2:
    statement2
elif condition3:
    statement3
...
else:
    defaultStatement
#some code after block
```

- Check **condition1**
 - If true, **statement1** happens **and** we leave this entire block
 - If false, check next condition
- Check **condition2**
 - If true, **statement2** happens **and** we leave this entire block
 - If false, check next condition
- ...
- If every condition was false, we go to **else** and **defaultStatement** happens

Example `if-elif` Statements

```
if score >= 90:  
    grade = "A"  
elif score >= 80:  
    grade = "B"  
elif score >= 70:  
    grade = "C"  
elif score >= 60:  
    grade = "D"  
else:  
    grade = "F"
```


What is the difference between these two?

```
if condition1:
    statement1
elif condition2:
    statement2
elif condition3:
    statement3
else:
    defaultStatement
```

```
if condition1:
    statement1
if condition2:
    statement2
if condition3:
    statement3
else:
    defaultStatement
```

Question

- Simple conditions are comparisons where exactly 2 values are involved
- What if we want more complicated conditions?



Is this ball
red **AND** round?



Is this ball
orange **AND**
round?

Compound Conditions

- Logical operators

not

and

or

- Combine simple conditions together with logical operators
- Logical operators combine 2 Boolean expressions
- Using compound conditions, we can make decisions based on how multiple groups of values compare

Pick a number between 1 and 10...

and

- Both must be true
 - Example:
 - Is your number greater than 5 **AND** less than 10?
-

or

- Either may be true
- Example:
 - Is your number 5 **OR** 10?

Syntax

and

expression1 and expression2

Both expression1 and expression2 must be True for the whole to be True

or

expression1 or expression2

Either expression1 or expression2 may be True for the whole to be True

Pick a number between 1 and 10...

and

- Is your number greater than 5 **AND** less than 10?
`if number > 5 and number < 10:`
-

or

- Is your number 5 **OR** 10?
`if number == 5 or number == 10:`

Syntax: **not**

- A Boolean expression can be negated using the **not** operator

- Syntax

not condition

- Examples

not (num >= 0)

a or b and not a and b

Try Writing Expressions...

Pick a number between 1 and 10...

- Is your number between 3 and 7?
- Is your number smaller than 5 or greater than 10?
- Is your number odd?

Try Writing Expressions...

Pick a number between 1 and 10...

- Is your number between 3 and 7?
if **number > 3 and number < 7**
- Is your number smaller than 5 or greater than 10?
if **number < 5 or number > 10**
- Is your number odd?
if **((number % 2) == 1)**
if **((number % 2) != 0)**
if **not ((number % 2) != 1)**

Parentheses added for ease of reading

Truth Table

A	B	A and B	A or B	not A
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Importing Modules

- Modules are files that contain code meant to be used in other programs
- Python comes in many built-in modules
- We can use a module in our program by using the **import** command
- Syntax: **import moduleName**

Example

```
# Miles Morales
```

```
# ITP 115
```

```
# Assignment 5
```

```
import someModuleName
```

Place all **import** commands
right beneath the comment
header with your name

```
name = input("Enter your name: ")
```

```
...
```

Accessing Functions inside Modules

- Most modules will have functions (commands)
- We use these functions just like **print** and **input** with one difference
- To access a function inside a module, you must use the module name
- Syntax

moduleName.functionName()

Random Module

- Has useful functions to generate random numbers and produce random results
- **randrange(...)** is a function which produces a random integer
- Given an integer input, **randrange(...)** will select a random number going from **0** up to *but not including* that integer

Example: randrange

```
num = random.randrange(6)
```

- The variable **num** will store an integer randomly selected from a group of 6 numbers starting at 0

0, 1, 2, 3, 4, 5

```
num = random.randrange(21)
```

- The variable **num** will store an integer randomly selected from a group of 21 numbers starting at 0

0, 1, 2, 3, 4, 5, ... 18, 19, 20

Different Ranges

- What if you want numbers from 1-6, not 0-5?
- **Shift the range!**

random.randrange(6) + 1

- Why does this work?

Why does this work?

`random.randrange(6)+1`

Random Number
0
1
2
3
4
5

+1

=

Result
1
2
3
4
5
6

Math Module

- The **math** module contains basic mathematical functions
- Some functions are **sqrt**, **tanh**, **sin**
- Ex

```
num = math.sqrt(16)  
print(num)
```

4

Popular Libraries and Their Uses

- NumPy:
 - Complex math
- Pandas
 - Data analysis
- Request:
 - Connecting to data on the web
- Matplotlib
 - generating graphs
- PyQt / Tkinter
 - creating graphical interfaces

FOR REFERENCE ONLY

Evaluating Any Value as True or False

- Any value in Python can be evaluated as either **True** or **False**
- So **2749**, **8.6**, **0**, **"banana"**, and **""** can each be evaluated as **True** or **False**
- It may seem bizarre, but this is valid in Python and can sometimes make for more elegant conditions

Rules for Evaluating Any Value as True or False

- Numbers
 - 0 and 0.0 are **False**
 - All other numbers positive and negative are **True**
- String
 - The empty string "" is **False**
 - Everything else is **True**
- Other variables (*later in semester*)
 - Anything that is considered *empty* is **False**
 - Everything else is **True**

Testing for
empty is very
common



Treating Values as Conditions

```
mystery = "chicken"  
if mystery:  
    print("Condition is true")  
Condition is true
```

Treating Values as Conditions

```
mystery = 0
if mystery:
    print("Condition is true")
else:
    print("Condition is false")
```

Condition is false

Understanding True and False

- What type of variable is **x**?

x = False

- What type of variable is **y**?

y = "False"

- Is the following considered **True** or **False**?

if x:

- Is the following considered **True** or **False**?

if y: