AI Boot Camp

Programming Decisions

Module 2 Day 2

© 2023 edX Boot Camps LLC. Confidential and Proprietary. All Rights Reserved.

Class Objectives

By the end of this class, you will be able to:

1. Construct lists and tuples, manipulate list elements, and apply list functions.
2. Formulate If-else statements, apply comparison operators, and validate if a string is a number.
3. Integrate `elif` statements for more complex decision-making, utilize operators effectively, and employ the `type()` function for input validation.
4. Implement for loops and while loops.
5. Develop nested conditionals and nested loops and use the break statement to interrupt loop execution.

Instructor **Demonstration**

Lists and Tuples

Data structures are anything that holds information and **isn't** a variable. In fact they often store **multiple variables.**

Lists

A few points to keep in mind:

1. Lists hold multiple pieces of data within one variable.

2. Lists can hold multiple types of data, such as strings, integers, and Boolean values, all within a single list.

Indexing and Slicing

The position of items in a given list is referred to as the **index.**

- Indexes start at 0, so the first item in a list will be at position 0 and the second at position 1.
- To reference a list item from the end of the list, the index begins with a minus symbol.
- You can call a particular item out of a list with it's index:
  - i.e. `info_list[0]` or `info_list[-1]`

Slicing calls parts of a list, from one index **up to** another.

This is done by including : in the list parameter

- Select items from the start index to the stop index.

  `Info_list[start:stop]`
- Select items from the start index to the end of the list.

  `Info_list[start:]`
- Select items up to the stop index.

  `Info_list[:stop]`
- Select all items in the list.

  `info_list[:]`

List Functions in Python

Python has a set of built-in methods that you can use on lists:

`index`	method returns the index of a list item.
`append`	method adds elements to the end of a list.
`pop`	method can remove a value by index.
`remove`	method deletes a given value from a list.
`len`	function returns the length of a list.
`max`	returns the maximum value in the list.
`min`	returns the minimum value in the list.
`sum`	adds all numbers in the list together.

Tuples

Tuples are functionally similar to lists in what they can store, but they are immutable.

1. While lists in Python can be modified after their creation, tuples can never be modified after their declaration.

2. Tuples tend to be more efficient to navigate through than lists and also protect the data stored within from being changed.

```
# Creates a tuple, a sequence of immutable Python objects that cannot be changed
info_tuple = ('Python', 100, 4.65, False)
print(info_tuple)
```

Activity:
Create a Menu

Practice using tuples, lists, list functions, and methods to create and modify a restaurant menu.

Suggested Time:
15 Minutes

Time’s up!

Let’s review

Questions?

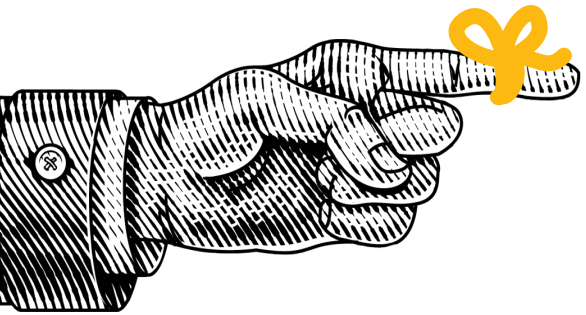
Instructor **Demonstration**

Conditionals

Conditionals

A few points to keep in mind:

1. Conditionals allow you to direct what actions a program should take under particular circumstances.



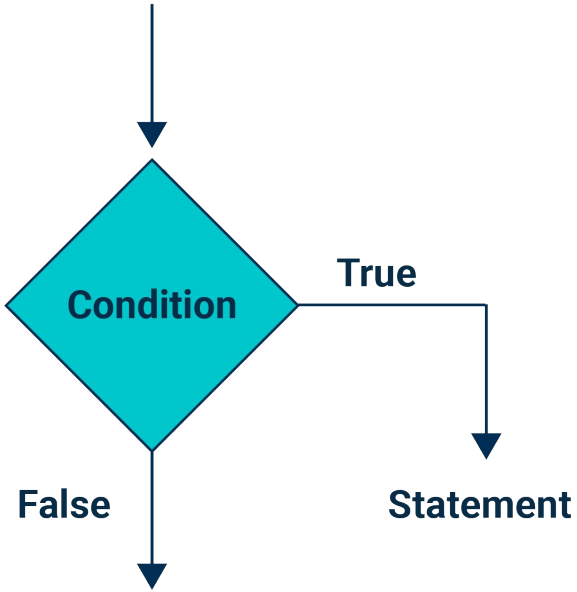
Remember the dish-packing pseudocode from the last lesson?

There we used conditionals to direct which dishes needed to go where.

Different Conditional Statements

1. `if`
2. `if-else`
3. `elif`

if

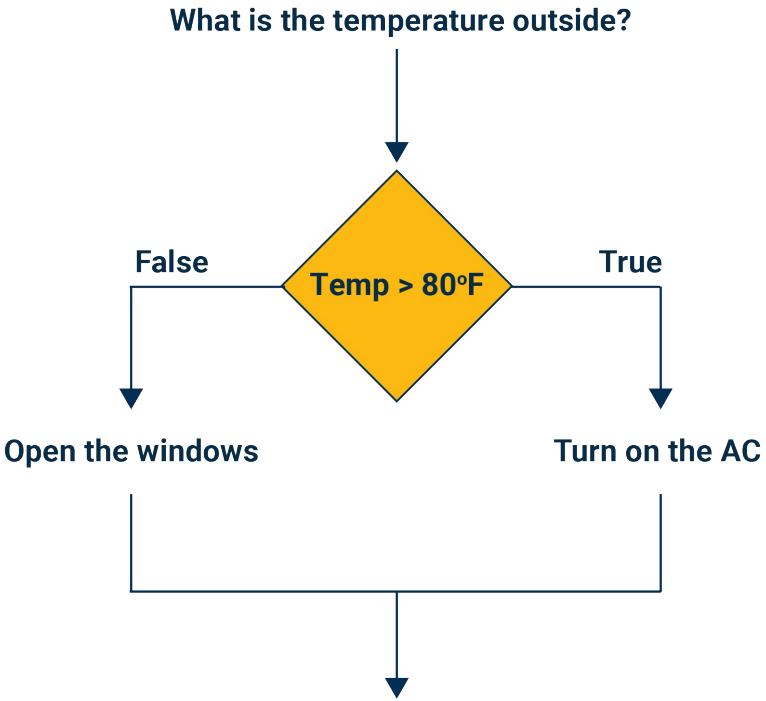



```
If condition:
     statement
```

For example:

```
if score >= 90:
     grade = 'A'
```

if-else




```
if condition:
     statement 1
     statement 2
else:
     statement 1
     statement 2
```

For example:

```
if temp > 80:
     turn_on_AC()
else:
     open_windows()
```

Comparison Operators

Operator x	Meaning
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	equal to
!=	not equal to

Input Validation

Input validation is a kind of conditional concerned with checking if a user's input is a valid input for the purpose it will be used for.

For example, if you want to cast a string as an integer, you should first check that the input is in fact a number before doing so to avoid problems down the line. We can use `.isdigit()` to achieve this:

```
if string.isdigit():
    number = int(string)
```

NOTE: Indentation is important!

Activity:

Conditional Conundrum

Review prewritten conditionals and predict the lines that will be printed to the console.

Suggested Time:
10 Minutes

Time’s up!

Let’s review

Questions?

Instructor **Demonstration**

Complex Conditionals

Logical Operators

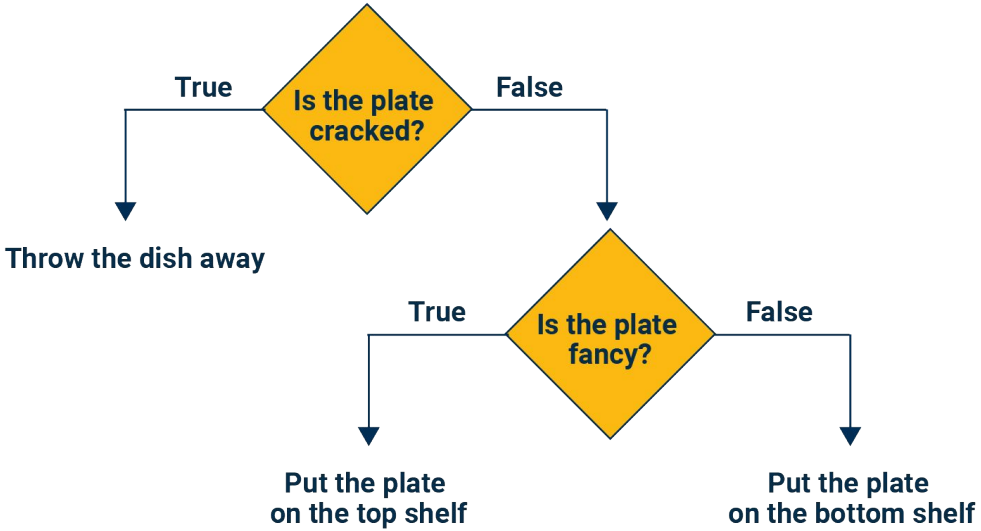
Operator	Meaning	Example

and	Evaluates two Boolean expressions into one compound expression. The compound expression is true if both Boolean expressions are true.
	
	If one of the expressions is false, then the compound expression is false.
		`x = 5`
		`y = 5`
		`if x == 5 and y == 5:`
		`  print("True")`
		`else:`
		`  print("False")`
		
		This prints "True" because *x* = 5 is true and *y* = 5 is true.

or	Evaluates two Boolean expressions into one compound expression. The compound expression is true if either Boolean expression is true.
	
	If one of the expressions is false, then the compound expression is true. If both expressions are false, then the compound expression is false.
		`x = 5`
		`y = 5`
		`if x == 3 or y == 5:`
		`  print("True")`
		`else:`
		`  print("False")`
		
		This prints `"True"` because *x* = 3 is false and *y* = 5 is true.

not	Evaluates a Boolean expression. The expression is true if the conditional is false.	`x = 5`
		`y = 5`
		`if not(x > y):`
		`  print("True")`
		`else:`
		`  print("False")`
		
		This prints `"True"` because *x* is not greater than *y*. If *x* = 6, then it would print `"False"` because *x* is greater than *y*.

elif




```
# Check multiple conditions
plate = "fancy"

if plate == "cracked":
    print("Throw the dish away")
elif plate == "fancy":
    print("Put the plate on the top
shelf")
else:
    print("Put the plate on the
bottom shelf")
```

Membership Operators

Operator	Meaning	Example

in	Returns True if a sequence with the specified value is present in the object.	`counties = ["Arapahoe","Denver","Jefferson"]`
		`if "Arapahoe" in counties:`
		`    print("True")`
		`else:`
		`    print("False")`
		
		This prints `"True"` because *Arapahoe* is in the counties list.

not in	Returns True if a sequence with the specified value is not present in the object.	`counties = ["Arapahoe","Denver","Jefferson"]`
		`if "El Paso" not in counties:`
		`    print("True")`
		`else:`
		`    print("False")`
		
		This prints `"True"` because *El Paso* is not in the counties list.

Identity Operators		
Operator	**Meaning**	**Example**

is	Returns True if the two objects are the same object.	`if type(["Oceania", "Europe", "Asia"]) is list:`
		`  print("Object type is a list")`
		`else:`
		`  print("Object type is not a list")`
		
		This prints `"Object type is a list"` because the text inside the type() function is indeed a list.

is not	Returns True if the two objects are not the same object.	`if type("This is not an integer") is int:`
		`  print("True")`
		`else:`
		`  print("False")`
		
		This prints `"False"` because ”This is not an integer” is not an int

Activity:

Smooth Operator

Practice writing complex conditional statements with comparison, logical, membership, and identity operators.

Suggested Time:
15 Minutes

Time’s up!

Let’s review

Break

15 mins

Instructor **Demonstration**

Loops

Loops

Loops are used when a particular action needs to be repeated multiple times

- The variable `x` is created within the loop statement and could theoretically take on any name as long as it is unique.
- When looping through a range of numbers, Python will halt the loop one number before the final number. For example, when looping from 0 to 5, the code will run 5 times, but `x` will only ever be printed as 0 through 4.
- When provided with a single number, `range()` will always start the loop at 0. However, when provided with two numbers, the code will loop from the first number until it reaches one fewer than the second number.

```
# Loop through a range of numbers (0 through 4)
for x in range(5):
    print(x)

print("_____________________________________________________")

# # Loop through a range of numbers (2 through 6)
for x in range(2, 7):
    print(x)

print("_____________________________________________________")
```

Looping through Strings

Python can also loop through all the letters within a string.

The syntax is for `<variable> in <string>`:

```
# Iterate through letters in a string
word = "Peace"
for letters in word:
    print(letters)

print("_____________________________________________________")
```

Looping through Lists

Python can also loop through all the values within a list.

The syntax is for `<variable> in <list>`:

```
# Iterate through a list
zoo = ['cow', 'dog', 'bee', 'zebra']
for animal in zoo:
    print(animal)

print("_____________________________________________________")
```

`while` Loops

These are just like `for` loops but will continue looping for as long as a condition is met.

```
# Loop while a condition is being met
run = 'y'
while run == 'y':
    print('Hi!')
    run = input("To run again. Enter 'y'")
```

Special Operators for use in Loops

`+=`

Performs addition with the two items on either side of the operator, assigns the result to the left item.
Instead of typing out `x = x + y` you can simply use `x += y.`

`_`

Also known as a dummy variable.
It is useful as a placeholder in loops where you only need a simple counter that you will not be calling or using again at a later stage.

e.g., If you wanted to print the String “Hello!” four times:

```
for _ in range(4):
    print("Hello!")
```

Activity:

Number Chain — Loops

Revisit the travel scenario and practice using loops with lists.

Suggested Time:
15 Minutes

Time’s up!

Let’s review

Instructor **Demonstration**

Nested Decisions

Nested Decisions

Nesting is simply adding at least one additional layer, or depth, to the code.

A nested **`if-else`** statement is created by adding one or more other **`if-else`** statements inside the first one.

You can nest **`for`** or **`while`** loops in the same way.

Nested `if`:

```
if condition_1:
    if condition_2
        statement_1
    else:
        statement_2
else:
    statement_3
```

Nested `for`:

```
x = [1, 2]
y = [3, 4]

for i in x:
    for j in y:
        print(i, j)
```

Nested `if-else`

- The script first checks the value of `price`. If it is a negative number, it will skip everything in the if statement, without checking the value of `issue_currency`, and jump straight to performing the action inside the else statement.
- If the `price` is greater than or equal to 0, then it will reach the next layer of the nested conditional, and test the equality of `issue_currency` to determine what action to perform.
- The comments highlight what's happening at each step, and can be read like pseudocode.

```
# Check if price is not negative (greater than equal to 0)
if price >= 0:
    # If price is not negative and currency is 'USD'
(Dollar).
    if issue_currency == "USD":
        print("The currency is $", price)
    # If price is not negative and currency is 'EUR' (Euro).
    elif issue_currency == "EUR":
        print("The currency is €", price)
    # If anything other than the above.
    else:
        print("The currency is not in USD or EUR.")
# Else price is negative
else:
    print("Error, the price listed is a negative number")
```

Exiting a Loop Before The Loop's Condition is Met

A loop can be ended early with the `break` keyword.

Within the loop, write a conditional statement that should be met in order to exit the loop. For example:

```
if user_input == 'q':
    break
```

The `break` keyword only exits the first loop it is associated with. To exit a loop it is nested in, another conditional statement and `break` is required.

`break` allows us to exit a continuous loop where the condition is always true.

```
while True:
    print(x)
    if condition_met:
        break
```

Activity:

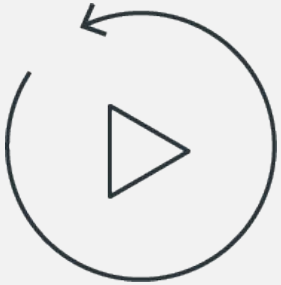
Loop de Loop

Practice using nested loops and nested conditionals while storing information about an amusement park's rides and prices.

Suggested Time:
20 Minutes

Time’s up!

Let’s review



Let’s **recap**

Recap

Today you learned about:

1. Lists.
2. `if-else` statements and nested conditionals.
3. Membership operators.
4. `for` and `while` loops (and nested loops).
5. Input validation with `isdigit()` and `type()`.

Next

Questions?

The End