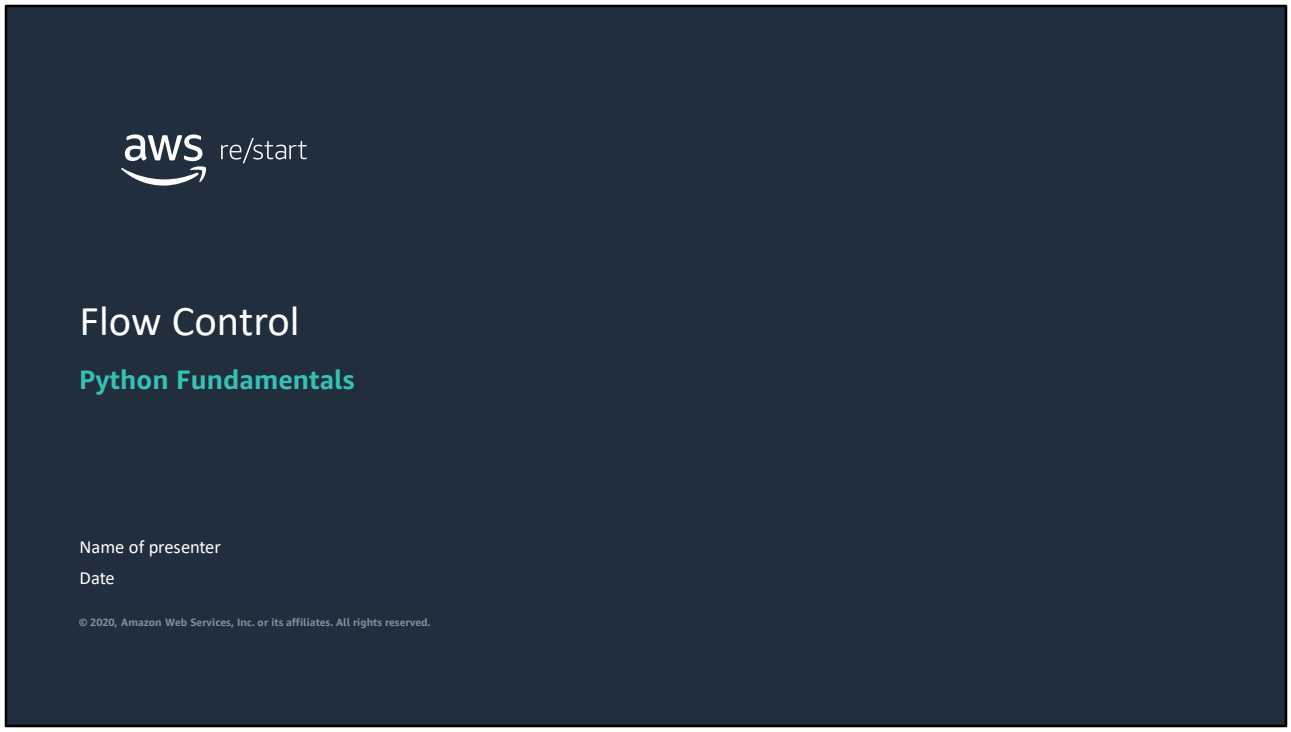


Welcome to Flow Control.

What you will learn

At the core of the lesson

You will learn how to:

- Describe the purpose and use of flow control in a program
- Use conditional statements to run code based on evaluated criteria
- Use loops to run code repeatedly until the criteria are met
- Use the input() function to prompt users for input to be used when they run the program

aws re/start

In this module, you will learn how to:

- Describe the purpose and use of flow control in a program
- Use conditional statements to run code based on evaluated criteria
- Use loops to run code repeatedly until the criteria are met

Flow control

Flow control is the order that individual statements, instructions, or function calls of an imperative program are run or evaluated.

Control refers to the order that the interpreter runs the code.

Different statements can control this order.

Sometimes, you want something to happen when a condition is met. Then, you use *conditional statements*.

If: The keyword ***if*** means that if this condition is met, complete this action.

Elif: The keword ***elif*** is short for *else if*. Else, if this condition is true, complete this other action. (You can have multiple ***elif*** statements in a single block.)

Else: The keyword ***else*** means that if none of the above conditions are met, complete this action.

Conditional statements: Examples

Some examples of how conditionals work in your everyday life are:

If the time is between these hours, the store is open. Else, the store is closed.

If the gas in the gas tank is below a certain level, turn on the low fuel warning.

If an employee has their badge, allow them into the building. Else, the door is locked.

If someone's age is greater than 18, consider them to be an adult. Else, if their age is between 13 and 18, consider them to be a teen. Else, consider them a child.

Conditionals in code

Consider the previous example of human age and life stages, and code it into Python. With conditionals, the indentation becomes important. You must group or nest the related conditionals together.

```
if age > 18:
...       print("adult")
else:
...       print("child")
```

You must tell Python which age you want it to evaluate. Create a variable that is called *age* and assign it a numerical value. You must do this step *before* you write your conditional statements.

Conditionals in code, continued

Your code might look like this example:

```
age = 21

if age >18:
        print("adult")
else:
        print("child")
```

Some things to note:

- The colons *after* the conditionals are important.
- You also use the **print** function. If you run this code, what do you think Python will print?

Demonstration: Conditionals

1. Create a new file that is called: **conditionals.py**
2. Create a set of conditionals using “if”, “elif”, and “else” statements in Python. They should be created so that:

   If there are 5 or more bananas print “I have a bunch of bananas.”

   if there are 1–4 bananas print “I have a small bunch of bananas.”

   If there are no bananas print “I have no bananas.”
3. Assign **print** statements to your conditions using the above rules.
4. Create a variable to hold the number of bananas.
5. Run the code to observe the result.

Code example:

```
bananas=1

if bananas >= 5:
        print("I have a large bunch of bananas”)
elif bananas >= 1:
        print(“I have a small bunch of bananas”)
else:
        print(“I don’t have any bananas”)
```

Loops

- Loops are a technique that tells Python to run a block of code repeatedly.
- Python runs loops *for* a certain number of times, or *while* a certain condition is true.
- The two types of Python loops are called **for** and **while**.

While loops

- **While** loops can run indefinitely, so you must include the condition for the loop to stop. Otherwise, the code creates an infinite loop.
- It is common to use an *iterative counter* with loops. As the loop completes, the counter increases (or decreases). When the counter reaches a specific number, the loop stops.

Example:

```
counter = 0

while counter <= 3:
        print("I love learning Python!")
        counter = counter + 1
```

Demonstration: While Loops

1. Create a new file that is called: **while.py**
2. Create a **while** loop with the code from the previous slide.
3. Assign **print** statements to your loop.
4. Run the code and observe the result.

Lists

```
>>> [1, 2, 3]
>>> ["alice", "bob"]
>>> [1, "alice", 2, "bob"]
>>> []
```

- Lists are a mutable data type. Lists can contain multiple data types (strings, ints, floats, and even other lists).
- Lists are denoted with brackets ([ ]) on each end.
- Values are enclosed in brackets, and they are separated with commas.
- Any number of items can be in a list—even zero (no) items.

Demonstration: Lists

1. Create a file that is called: **lists.py**
2. Create a list, and place three or more items in that list. Try mixing the data types, or making multiple lists.
3. Print your list.

aws re/start

Code demo:

```
class_roster = [“Xiulan","“Kwaku","“Shirley"]
test_scores = [86,93,80]

print(class_roster)
print(test_scores)
```

For loops

- A **for** loop reads: for each element in ***<thing>***, do a certain task.
- Some real-life examples are:

For every egg in a recipe, add 2 cups of flour.

For every package on our truck, add 2 kilograms.

For every hamburger that is ordered, subtract 1 from the inventory.

Loops and lists

```
for num in [1, 2, 3]:
    ...
    print(num)
```

- **For** loops and lists work well together.
- For every item in this list, Python prints that item.
  - Every time it calls **num**, it assigns a value from the list (1, then 2, . . .) to **num**.
  - The loop then prints the value.
  - After it goes through the entire list of values, the loop stops.

Demonstration: For Loops

1. Open your **lists.py** file.
2. Create a **for** loop that prints your list.
3. Run the code and observe the result.

aws re/start

Code demo:

```
class_roster = [“Xiulan",“Kwaku",“Shirley"]
test_scores = [86,93,80]

for student in class_roster:
        print(student)

for score in test_scores:
        print(score)
```

Dictionaries

```
>>> {"key": "value"}
>>> {0:100, 1:200, 2:999}
>>> {0:{1:2}}
>>> myDict = {}
```

- Dictionaries contain immutable keys, which are associated to their values. Keys must be immutable data types.
- Dictionaries can be nested inside each other.
- To create an empty dictionary, use a pair of braces with nothing inside: `{}`
- Keys are separated from their values with a colon: `{"Key":"Value"}`
- Retrieve a value in the dictionary by its key: `myDict.get("key") or myDict["key"]`

Demonstration: Dictionaries

1. Create a file that is called **dictionary.py.**
2. Create a dictionary and create four ***key:value*** pairs in it.
3. Print the third value in your dictionary.
4. Print your entire dictionary.

Code example:

```
myDict = {}
myDict["one"] = 1
myDict["two"] = 2
myDict[3] = "three"
myDict["four"] = 4.4

print(myDict[3])

for key, value in myDict.items():
        print(key, value)
```

Input

```
>>> response =
input("What's your name? ")
What's your name? Jorge
>>> response
'Jorge'
```

- The **input()** function asks the user to enter text and saves the result to a variable.
- One optional argument is a prompt for the user, as a string.

Demonstration: “Hello, Your Name”

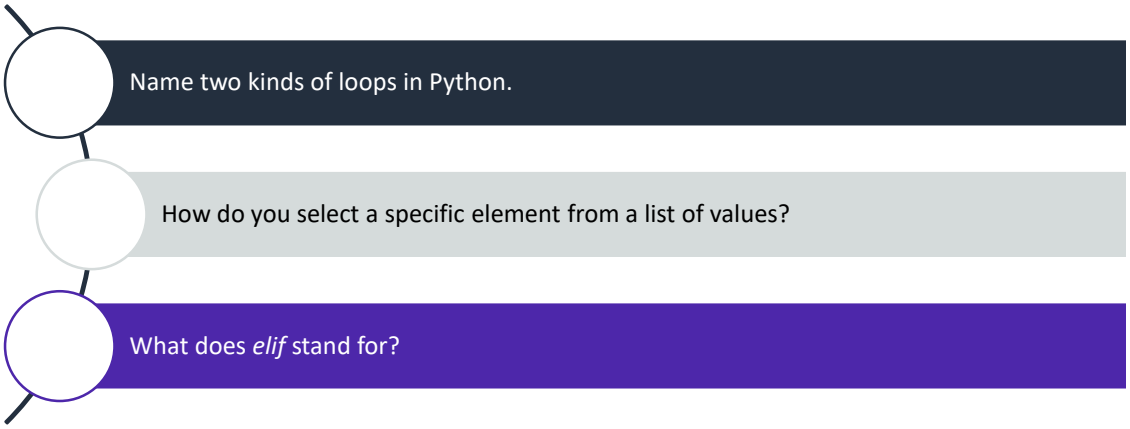
1. Open **helloworld.py**.
2. Get input from user.
3. Print the response.
4. Can you concatenate the input so that the output prints **hello <*response*>**?
5. Run the program again.

aws re/start

Code example:

```
name = input("What’s your name?")
print("Hello " + name)
```

Checkpoint questions



aws re/start

Answers:

1. Two kinds of loop flow structures in Python are *for* and *while*.
2. Specific values in a list are specified by an index in brackets ([ ]) after the list name. For example, *myList[2]* selects the third element in the *myList* variable.
3. The statement *elif* follows an *if* statement. If the preceding *if* or *elif* conditional statement evaluates to false, then the *elif* is evaluated.

Key takeaways

- An *if* conditional statement evaluates a condition and, if *true*, it runs a block of code.
  - *elif* and *else* statements follow *if* statements
- A *while* loop runs a code block until a condition returns *false*.
- A list is a collection of values in a specific order. The values do not have to be the same type.
- A dictionary is a collection of key-value pairs, where the value is accessed by using the key.

aws re/start

Some key takeaways from this lesson include:

- An *if* conditional statement evaluates a condition and, if *true*, runs a block of code.
  - *elif* and *else* statements follow *if* statements
- A *while* loop runs a code block until a condition returns *false*.
- A list is a collection of values in a specific order. The values do not have to be the same type.
- A dictionary is a collection of key-value pairs, where the value is accessed by using the key.