Conditional statements and operators

INTRODUCTION TO PYTHON FOR DEVELOPERS

George Boorman
Curriculum Manager, DataCamp

datacamp

Booleans

```
# Boolean variable
the_truth = True
print(the_truth)
```

```
True
```

- Used to make comparisons

Operators

- Comparison operators
  - Symbols or combinations of symbols
  - Used to compare things
  - Similar to symbols for calculations such as `*`, `+`, `-` etc.
- Check if two things are equal
  - `==`

Checking for equality

```
# Compare if 2 is equal to 3
2 == 3
```

```
False
```

```
# Check that 2 is not equal to 3
2 != 3
```

```
True
```

- Common use-case: checking login details

Numeric comparison operators

```
# Is 5 less than 7?
5 < 7
```

```
True
```

```
# Is 5 greater than 7?
5 > 7
```

```
False
```

```
# Is 5 less than or equal to 7?
5 <= 7
```

```
True
```

```
# Is 5 greater or equal to 7?
5 >= 7
```

```
False
```

Other comparisons

```
# Is James greater than Brian
"James" > "Brian"
```

```
True
```

- Strings are evaluated in alphabetical order

Conditional statements

- If `True` perform a task
  - otherwise, do nothing
- `if` > condition is met > perform action

```
# Target units sold and actual units sold
sales_target = 350
units_sold = 355

# Compare sales
if units_sold >= sales_target
```

Conditional statements

```
# Target units sold and actual units sold
sales_target = 350
units_sold = 355

# Compare sales
if units_sold >= sales_target:
```

Conditional statements

```
# Target units sold and actual units sold
sales_target = 350
units_sold = 355

# Compare sales
if units_sold >= sales_target:
    print("Target achieved")
```

```
'Target achieved'
```

Indentation

```
# Target units sold and actual units sold
sales_target = 350
units_sold = 355

# Compare sales
if units_sold >= sales_target:
print("Target achieved") # This line is not indented
```

```
    print("Target achieved")
    ^
IndentationError: expected an indented block
```

Elif statement

```
# Target units sold and actual units sold
sales_target = 350
units_sold = 325
# Compare sales
if units_sold >= sales_target:
    print("Target achieved")
# Check if we were close to the target
elif units_sold >= 320:
    print("Target almost achieved")
```

- Can use as many `elif` keywords as we like!

Else statement

```
# Compare sales
if units_sold >= sales_target:
    print("Target achieved")
# Check if we were close to the target
elif units_sold >= 320:
    print("Target almost achieved")
# Otherwise...
else:
    print("Target not achieved")
```

Comparison operators cheat sheet

Operator	Function
`==`	Equal to
`!=`	Not equal to
`>`	More than
`>=`	More than or equal to
`<`	Less than
`<=`	Less than or equal to

Keyword	Function	Use
`if`	If condition is met	First in the workflow
`elif`	Else check if condition is met	After `if`
`else`	Else perform this action	After `elif`

Let's practice!

INTRODUCTION TO PYTHON FOR DEVELOPERS

For loops

INTRODUCTION TO PYTHON FOR DEVELOPERS

George Boorman
Curriculum Manager, DataCamp

datacamp

Individual comparisons

```
# Prices list
prices = [9.99, 8.99, 35.25, 1.50, 5.75]
prices[0] > 10
```

```
False
```

```
prices[0] < 5
```

```
False
```

```
prices[0] >= 5 and prices[0] <= 10
```

```
True
```

For loop syntax

```
for value in sequence:
    action
```

- `for` each `value` in `sequence`, perform `action`
  - `action` is indented because of the colon in the previous line
- `sequence` = iterable e.g., list, dictionary, etc
- `value` = iterator, i.e., the index
  - Placeholder (can give it any name), `i` is common.

Print individual values

```
# Prices list
prices = [9.99, 8.99, 35.25, 1.50, 5.75]

# Print each value in prices
for price in prices:
    print(price)
```

```
9.99
8.99
35.25
1.5
5.75
```

Conditional statements in for loops

```
for price in prices:
```

Conditional statements in for loops

```
for price in prices:
    # Check if the price is more than 10
    if price > 10:
```

Conditional statements in for loops

```
for price in prices:
    # Check if the price is more than 10
    if price > 10:
        print("More than $10")
```

Conditional statements in for loops

```
for price in prices:
    # Check if the price is more than 10
    if price > 10:
        print("More than $10")
    # Check if the price is less than 5
    elif price < 5:
        print("Less than $5")
```

Conditional statements in for loops

```
for price in prices:
    # Check if the price is more than 10
    if price > 10:
        print("More than $10")
    # Check if the price is less than 5
    elif price < 5:
        print("Less than $5")
    # Otherwise print the price
    else:
        print(price)
```

Conditional statements in for loops

```
9.99
8.99
More than $10
Less than $5
5.75
```

Looping through strings

```
username = "george_dc"
# Loop through username and print each character
for char in username:
    print(char)
```

```
g
e
o
r
g
e
_
d
c
```

Looping through dictionaries

```
products_dict = {"AG32":87.99, "HT91":21.50,
                 "PL65":43.75, "OS31":19.99,
                 "KB07":62.95, "TR48":98.0}

# Loop through keys and values
for key, val in products_dict.items():
    print(key, val)
```

```
AG32 87.99
HT91 21.5
PL65 43.75
OS31 19.99
KB07 62.95
TR48 98.0
```

Looping through dictionaries

```
# Loop through keys
for key in products_dict.keys():
    print(key)
```

```
AG32
HT91
PL65
OS31
KB07
TR48
```

```
# Loop through values
for val in products_dict.values():
    print(val)
```

```
87.99
21.5
43.75
19.99
62.95
98.0
```

Range

- Can use `for` loops to update variables

```
range(start, end + 1)
```

- `start` = inclusive
- `end` = not inclusive

```
for i in range(1, 6):
    print(i)
```

```
1
2
3
4
5
```

Building a counter

```
# No visits yet
visits = 0
# Loop through numbers 1-10
for i in range(1, 11):
    # Add one to visits during each iteration
    visits += 1 # Same as visits = visits + 1

print(visits)
```

```
10
```

Let's practice!

INTRODUCTION TO PYTHON FOR DEVELOPERS

While loops

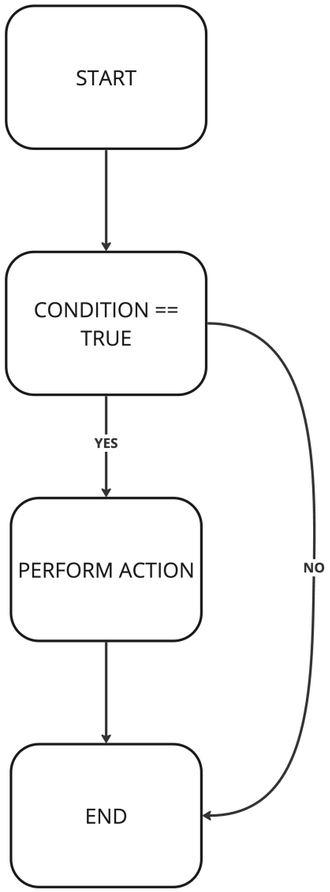
INTRODUCTION TO PYTHON FOR DEVELOPERS

George Boorman
Curriculum Manager, DataCamp

datacamp

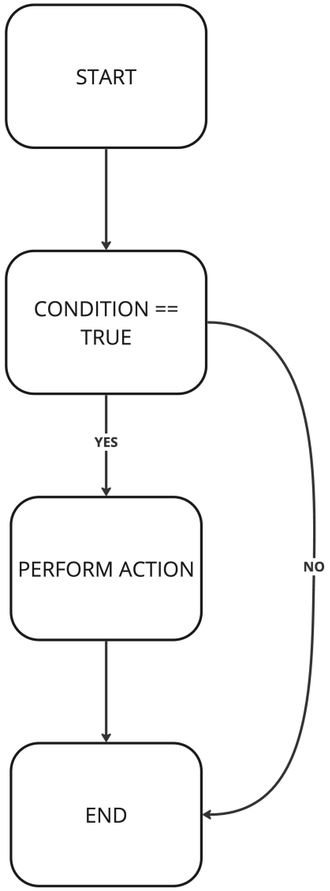
If statement

If statement

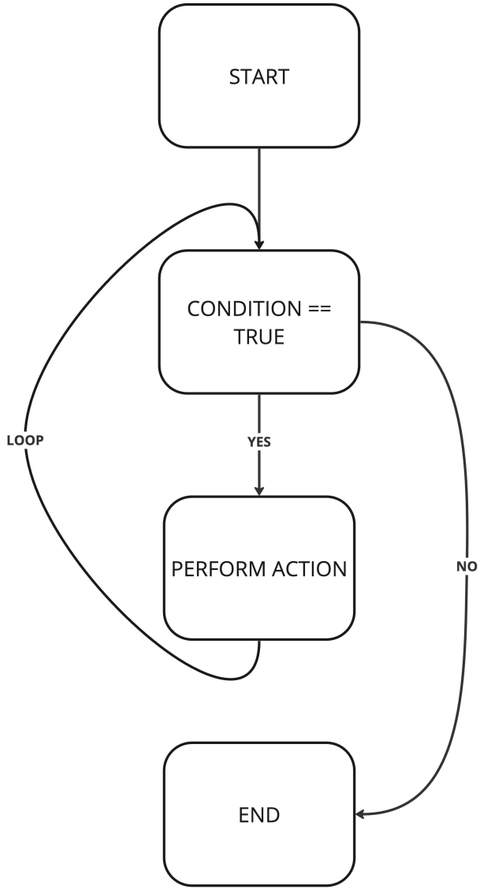


If statement versus while loop

If statement



While loop



While loop

```
while condition:
    action
```

- Any continuous task
  - Accelerate `while` a button is pressed
  - Monitor `while` below/above a threshold

[1] https://unsplash.com/@joaoscferrao

While loop

```
# Stock limit
stock = 10

# Number of purchases
num_purchases = 0

# While num_purchases < stock limit
while num_purchases < stock:
    # Increment num_purchases
    num_purchases += 1
    # Print remaining stock
    print(stock - num_purchases)
```

Output

```
9
8
7
6
5
4
3
2
1
0
```

A word of caution

- `while` runs continually while the condition is met

```
# Stock limit
stock = 10
# Number of purchases
num_purchases = 0

# While num_purchases < threshold
while num_purchases < stock:

    # Print remaining stock
    print(stock - num_purchases)
```

Running forever

```
10
10
10
10
10
10
10
10
10
10
10
10
10
10
```

Breaking a loop

```
# While num_purchases < threshold
while num_purchases < stock:

    # Print remaining stock
    print(stock - num_purchases)

    # Terminate the loop
    break
```

- `break` can also be used in `for` loops
- If the code is already running: Control + C / Command + C

Conditional statements within while loops

```
# While num_purchases < threshold
while num_purchases < stock:
    # Increment num_purchases
    num_purchases += 1
    # Conditional statement inside the loop
    if stock - num_purchases > 7:
        print("Plenty of stock remaining")
    elif stock - num_purchases > 3:
        print("Some stock remaining")
    elif stock - num_purchases != 0:
        print("Low stock!")
    else:
        print("No stock!")
```

Conditional statements output

```
Plenty of stock remaining
Plenty of stock remaining
Some stock remaining
Some stock remaining
Some stock remaining
Some stock remaining
Low stock!
Low stock!
Low stock!
No stock!
```

Let's practice!

INTRODUCTION TO PYTHON FOR DEVELOPERS

Building a workflow

INTRODUCTION TO PYTHON FOR DEVELOPERS

George Boorman
Curriculum Manager, DataCamp

datacamp

Complex workflows

- Loops through data structures
  - `for`, `while`
- Evaluate multiple conditions
  - `if`, `elif`, `else`, `>`, `>=`, `<`, `<=`, `==`, `!=`
- Update variables
  - `+=`
- Return outputs
  - `print()`

The "in" keyword

- `in` = check if a value is **in** a variable/data structure

```
products_dict = {"AG32": 10, "HT91": 20,
                 "PL65": 30, "OS31": 15,
                 "KB07": 25, "TR48": 35}

# Check if "OS31" is a key in products_dict
if "OS31" in products_dict.keys():
    print(True)
else:
    print(False)
```

```
True
```

The "not" keyword

- `not` = check if a condition **is not** met

```
# Check if "OS31" is not a key in products_dict
if "OS31" not in products_dict.keys():
    print(False)
else:
    print(True)
```

```
True
```

The "and" keyword

- `and` = check if **multiple conditions** are met

```
# Check if "HT91" is a key and the minimum price of all products is > 5
if "HT91" in products_dict.keys() and min(products_dict.values()) > 5:
    print(True)
else:
    print(False)
```

```
True
```

The "or" keyword

- `or` = check if one (**or more**) condition is met

```
# Check if "HT91" is a key or that the minimum price of all products is < 5
if "HT91" in products_dict.keys() or min(products_dict.values()) < 5:
    print(True)
else:
    print(False)
```

```
True
```

Adding/subtracting from variables

- Combine keywords with other techniques to build complex workflows

```
sales_count = 0
for sale in range(1, 10):
    # sales_count = sales_count + 1
    sales_count += 1
stock = 10
for sale in range(1, 10):
    # sales_count = sales_count - 1
    stock -= 1
```

- Other ways to update variables

Appending

- Store information that meets specific criteria in a list

```
# Create an empty list
expensive_products = []

# Loop through the dictionary
for key, val in products_dict.items():

    # Check if price is 20 dollars or more
    if val >= 20:

        # Append the product ID to the list
        expensive_products.append(key)
```

Appending

```
print(expensive_products)
```

```
['HT91', 'PL65', 'KB07', 'TR48']
```

Let's practice!

INTRODUCTION TO PYTHON FOR DEVELOPERS

Congratulations!

INTRODUCTION TO PYTHON FOR DEVELOPERS

George Boorman
Curriculum Manager, DataCamp

datacamp

Chapter 1 recap

Syntax	Action	Example	Output
`*`	Multiply	`4 * 10`	`40`
`+`	Addition	`7 + 9`	`16`
`-`	Subtract	`23 - 4`	`19`
`/`	Division	`27 / 3`	`9`
`**`	Power	`3 ** 2`	`9`
`%`	Modulo	`7 % 4`	`3`

```
# Define total_spend
total_spend = 3150.96

# Single quotes
customer_name = 'George Boorman'

# Double quotes also work
customer_name = "George Boorman"
```

Chapter 2 recap

```
current_top_album = "For All The Dogs"

# Convert to lowercase
current_top_album = current_top_album.lower()
```

```
# List of prices
prices = [10, 20, 30, 15, 25, 35]
# Get the value at the first index
prices[0]
```

```
# Creating a dictionary
products_dict = {"AG32": 10, "HT91": 20,
                 "PL65": 30, "OS31": 15,
                 "KB07": 25, "TR48": 35}
```

```
# Create a prices set
prices_set = {10, 20, 30,
              15, 25, 35}
# Create a prices tuple
prices_tuple = (10, 20, 30,
                15, 25, 35)
```

Chapter 3 recap

Operator	Function
`==`	Equal to
`!=`	Not equal to
`>`	More than
`>=`	More than or equal to
`<`	Less than
`<=`	Less than or equal to

Keyword	Function	Use
`if`	If condition is met	First in the workflow
`elif`	Else check if condition is met	After `if`
`else`	Else perform this action	After `elif`

Chapter 3 recap

```
# Prices list
prices = [9.99, 8.99, 35.25, 1.50, 5.75]

# Print each value in prices
for price in prices:
    print(price)
```

```
9.99
8.99
35.25
1.5
5.75
```

Chapter 3 recap

```
# Stock limit
stock = 10
# Number of purchases
num_purchases = 0

# While num_purchases < stock limit
while num_purchases < stock:
    # Increment num_purchases
    num_purchases += 1
    # Print remaining stock
    print(stock - num_purchases)
```

Chapter 3 recap

Keyword	Function
`and`	Evaluate if multiple conditions are true
`or`	Evaluate one or more conditions are true
`in`	Evaluate if a value is in a data structure
`not`	Evaluate if a value is not in a data structure

- `list.append()`

Next steps

- Additional built-in functions
  - `zip()`
  - `enumerate()`
- Packages and modules
  - `os`
  - `time`
  - `venv`
  - `pandas`
  - `requests`
  - `numpy`
- Building custom functions

Congratulations!

INTRODUCTION TO PYTHON FOR DEVELOPERS