Python Fundamentals

Comments, code standards & conventions, logical and comparison operators, decisions, and math module

Objectives:

- What is a comment?
  - Single line comment
  - Multiline comment
- Code standards & conventions
  - Naming conventions
  - Indentation
- Comparison Operators
- Making decisions
- Logical operators

What is a comment?

- Comments are text that a programmer writes in a source file that will not be compiled/interpreted.
- Comments are usually added with the purpose of making the source code easier to understand.
- Lines of code can also be temporarily changed into comments so that the compiler/interpreter ignores them.

How to insert comments in Python

- Single Line: The simplest comment in Python. It starts with a hash character and continues to the end of the line.

```
number = 1 # set number to 1
```

- Multiline: Python has no multi-line commenting, per se. Instead you can use a hash character for each line.

```
# setting the
# age of the employee
age = 25
```

Code Standards & Conventions

- Different Python programmers can have different styles and approaches to the way they write programs. But by using standard Python naming conventions they make their code easier to read for themselves and for other programmers.

- In Python we use naming conventions for classes, variables, constants, etc.

Naming Styles

- There are many different naming styles. The following are the most commonly known ones.

- **snake_case** is where all the letters in a word are written in lower case. Each new word is separated by an underscore. (e.g. `customer_account`)

- **UPPER_CASE** is similar to **snake_case**, but all letters are upper case. (e.g. `MAX_HOURS`, `FIRST_DAY_OF_WEEK`)

Naming Styles

- **PascalCase** (also known as upper camel case) is where each new word begins with a capital letter. NO whitespaces are used. (e.g. `CustomerAccount`, `PlayingCard`)

- **camelCase** (also known as lower camel case) is the same as **PascalCase** except the first letter of the first word is in lowercase. (e.g. `hasChildren`, `customerFirstName`, `customerLastName`)

Code Standards & Conventions

- Classes should be **PascalCase**. Try to use nouns, because a class is normally representing something in the real world:

```
class Customer
class BankAccount
```

- Variables names should be **snake_case**. The names should represent what value is being stored:

```
first_name = "John"
order_number = 1
```

Code Standards & Conventions

- Constants are variables that should not be changed from their initial value. As this is just a convention, the Python style guide recommends upper cased snake case for constants. Making it clear the value should not be modified.

```
// Upper case usage
DEFAULT_WIDTH = 100
MAX_HEIGHT = 200
```

Indentation

- When writing code in any language, a commonly accepted practice is to indent some lines to improve readability.
- This is often encouraged, rather than required. Python is different.
- Python places a significance to indentation. Indentation is when a line of code is indented 2 or 4 spaces relative to the previous line.

Planning Decision-Making

- Decision structure
  - Involves choosing among alternative courses of action
  - Ask a question; the answer determines what path our code takes
- All computer decisions are yes-or-no decisions (true/false)
  - The Boolean values `True` and `False` values are used in every computer decision.
  - These decisions are called **Boolean Expressions**
- Generally, **Boolean Expressions** in Python involve comparison operators.

What is a comparison operator?

- A comparison operator is a **binary operator** that takes two operands whose values are being compared. They form the key to program control flow, known as conditional logic.
- Comparison operators used in Python:

```
< less than
> greater than
<= less than or equal to
>= greater than or equal to
== equal to
!= not equal to
```

What is a comparison operator?

- The result of an expression that involves comparison operators is always a bool value (true or false).

```
# For example:

x = 5
w = 7
result = x > w
```

in this case the value assigned to variable `result` would be `False`
(5 is not greater than 7)

```
# Another example:

x = 5
w = 7
result = (x + 2 == w)
```

in this case the value assigned to variable `result` would be `True`
(5 + 2 is equals to 7)

Examples of conditions

condition	*meaning*
`(x < 7)`	is the value of x less than 7 ?
`(x >= y)`	is x greater than or equal to y ?
`(x != y + z)`	is x not equal to y + z ?
`(x == y)`	x is equal to y ?

Control Flow – Conditional Logic

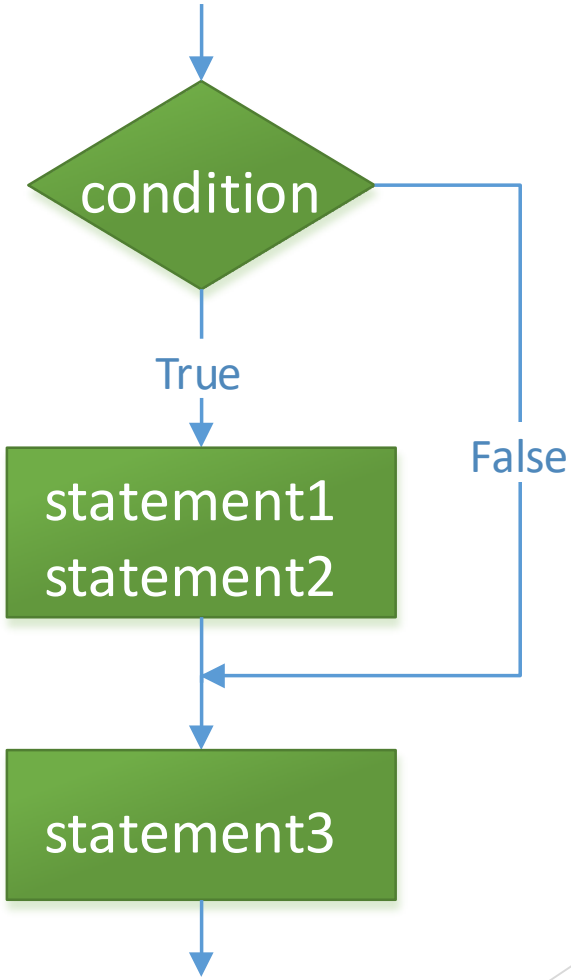
In a selection structure, a question is asked, and depending on the answer, the program takes one of two courses of action, after which the program moves on to the next instruction.

In Python we have three different types of selections:

- if
- if and else
- if and else if

if statement

```
if condition:
    statement1
    statement2
    …
statement3
```



if statement and indentation

- A quick note about the if statement. How does Python determine which statements belong to the if statement? With indentation.

- Statements that belong to an if statement **MUST** be indented. So in the previous slide, statements 1 and 2 belong to the if statement because they're indented 4 spaces relative to the if statement.

- Statement 3 DOES NOT, because it's NOT indented.

if statement

```
x = 1
if x > 10:
    x = x + 1
    x = x + 2
x = x + 3
print(x)
```

output?

```
x = 1
if x < 10:
    x = x + 1
    x = x + 2
x = x + 3
print(x)
```

output?

if statement

```
x = 1
if x > 10:
    x = x + 1
    x = x + 2
x = x + 3
print(x)
```

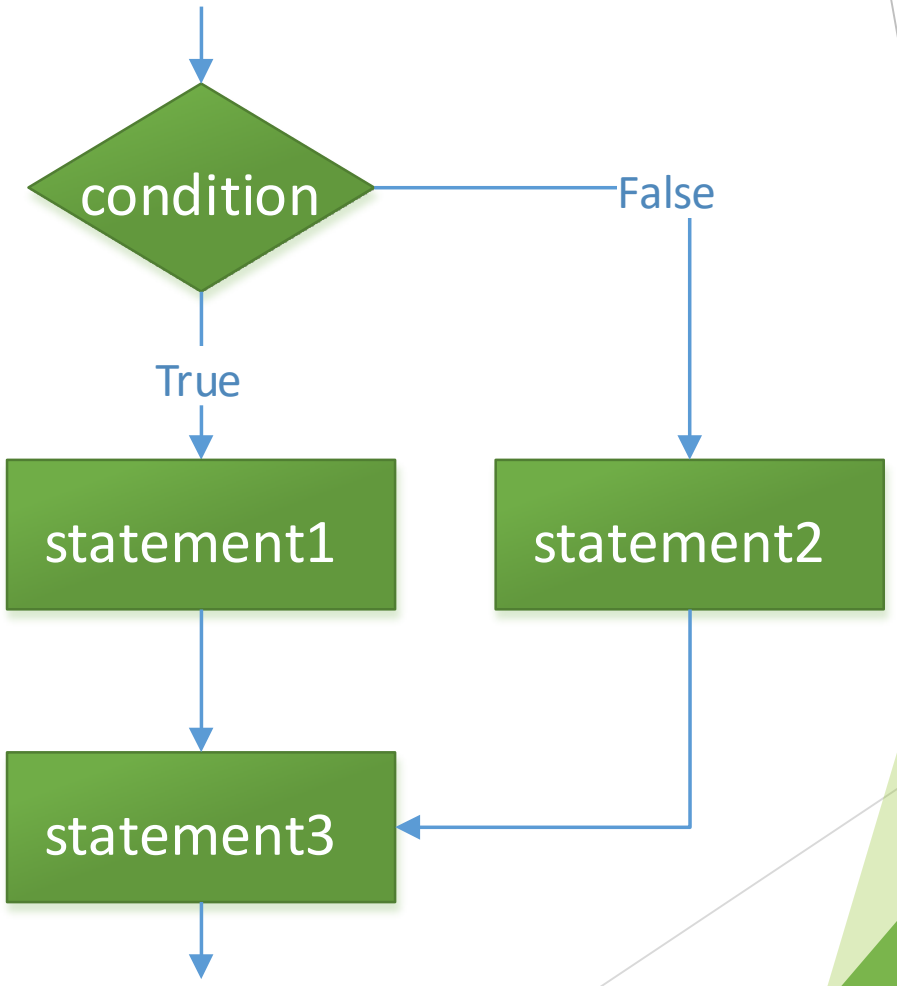
Output is 4:
1 is not greater than 10

```
x = 1
if x < 10:
    x = x + 1
    x = x + 2
x = x + 3
print(x)
```

Output is 7:
1 is less than 10

if statement with else

```
if condition:
    statement 1
    …
else:
    statement 2
    …
statement 3
```



Introduction to logical operators

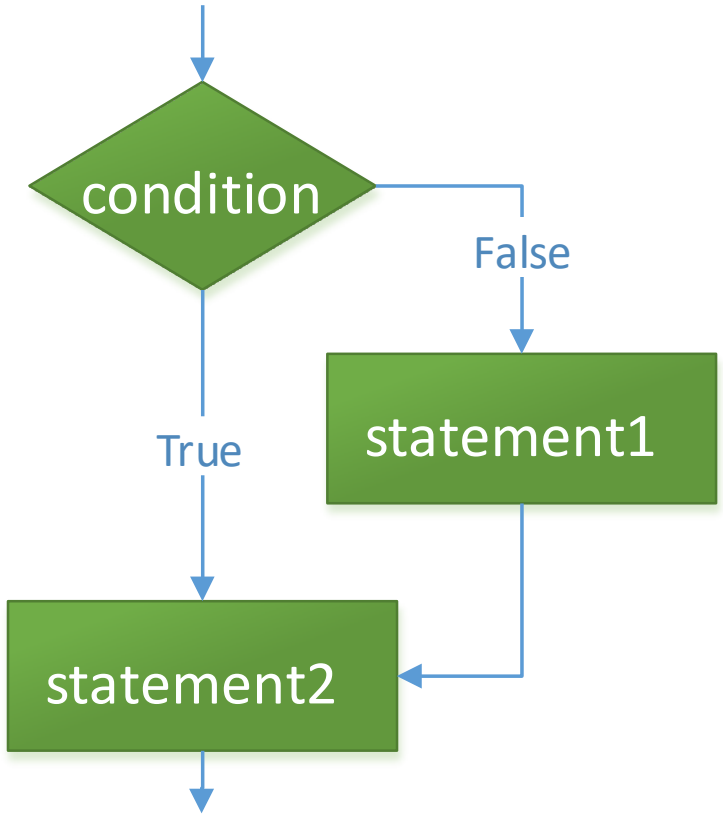
- There may be times where you want to test for a negative. For example, a number is NOT 5, or that the result of a comparison is NOT TRUE.

- In these situations you can use the logical operator `not`.

- For example, the expression `not 5 < 10` would evaluate to `False`. `5 < 10` evaluates to `True`, and the `not` operator turns the `True` into `False`.

if statement with else

```
if condition:
    pass
else:
    statement1
    …
statement2
```




```
if not condition:
    statement1
    …
statement2
```

if statement with else

```
if x > 10:
    pass
else:
    y = y + 1
```

```
if not x > 10:
    y = y + 1
```

```
if x <= 10:
    y = y + 1
```

These are all equivalent

Nested if statements

- When an `if` statement is contained inside another `if` statement, it's called a **nested if statement**.
- Used when two conditions must be met before some action is taken.
- Pay careful attention to the placement of `else` clauses.
- `else` statements are always associated with `if` based on their level of indentation. An `else` must be indented to the same amount as its associated `if` statement.

Nested if statements

- We want to create a small application in order to calculate the payment of an employee.

- If the employee sold more than 3 items during the week AND the total value for the items is greater or equal to $1250, then the employee will be awarded a $50 bonus.

Nested if statements

Below is our example code. Note how the nested if statement is indented 4 spaces relative to its containing if statement.

```
MIN_VALUE = 1250
MIN_ITEMS = 3
items_sold = 5
value_sold = 1300
if items_sold >= MIN_ITEMS:
    if value_sold >= MIN_VALUE:
        print("$50 bonus received")
```

More logical operators

As well as the **not** operator, there are two other logical operators. These are **and** and **or**.

The **and** operator allows us to construct Boolean expressions with multiple conditions, where each condition must be satisfied for the whole expression to evaluate to true.

Logical operator - and

Below we can see how the logic has been re-written to leverage the **and** operator.

```
MIN_VALUE = 1250
MIN_ITEMS = 3
items_sold = 5
value_sold = 1300
if items_sold >= MIN_ITEMS and value_sold >= MIN_VALUE:
    print("$50 bonus received")
```

Logical operator - or

- The logical operator **or** behaves similarly, in that it allows us to combine conditions. Except in the case of the **or** operator, only one condition has to be true for the whole expression to evaluate to true.

- Let’s change our scenario. If the employee sold more than 3 items during the week or the total value for the items is greater or equal to $1250, then the employee will be awarded a $50 bonus.

Logical operator - or

Below we can see how the logic has been re-written to leverage the **or** operator.

```
MIN_VALUE = 1250
MIN_ITEMS = 3
items_sold = 5
value_sold = 1300
if items_sold >= MIN_ITEMS or value_sold >= MIN_VALUE:
    print("$50 bonus received")
```

Functions

A function carries out some calculation or processing. It can be used without needing to know how this processing works.

A function usually needs to be given some values to work with. These are called **arguments**. You've already seen these with the `print` function.

A function often provides a value as the result of its processing. This is called a **return value**. You've already seen this with the `input` function.

Calling a function

To use a function, you "call" it by writing its name followed by arguments in parentheses.

A function name is always followed by parentheses, even if the function has no arguments.

Any word in Python code that is immediately followed by an opening parenthesis is a function name.

pow function

pow() is a function that calculates a number to the power of another. It has two arguments, both numbers.
It returns a numeric value.

When a function has several arguments they are separated by commas.

pow function

```
x = pow(2, 5)
```

`pow(2, 5)` is the function call.

The arguments are 2 and 5.

The function will calculate and return the value of 2^5.

This value (32) will then be assigned to x.

pow operator

```
x = 2 ** 5
```

Instead of using the pow function, you can use the power operator.
It uses two asterisks, and is equivalent to calling the pow function.

pow function and pow operator

Below are some more examples using either the pow function, or the pow operator.

```
z = pow(x, y)

area = pi * radius ** 2

volume = 4 / 3 * pi * pow(radius, 3)
```

Generating random numbers

The **random** module provides functions necessary to generate a random number. The **randint** function will generate a random integer within a given range.

```
import random
y = random.randint(0, 10)
```

y will have some value between 0 and 10

Introduction to modules

- In examining how to generate a random number, we were introduced to a new concept in Python; a *module*.

- Some functions, like `pow`, `print`, and `input` are available all the time. Others live inside of a module.

- We access these functions by *importing* the module. We then call a function from that module by preceding the function name with the name of the module.

math module

- While the `pow` function is available all the time, most mathematical functions exist in their own module.
- It shouldn't come as much surprise that this is called the `math` module.
- One such function is the `sqrt` function, which provides the ability to find out the square root of a number.

sqrt function

```
import math
x = math.sqrt(3)
```

math.sqrt(3) is the function call.
3 is the argument.
The function will calculate the square root of 3 and return the value.
This value will then be assigned to x.

References

Python style guide

https://www.python.org/dev/peps/pep-0008/

if statement

https://docs.python.org/tutorial/controlflow.html

References

random module documentation

https://docs.python.org/library/random.html

math module documentation

https://docs.python.org/library/math.html

Demonstration:

- What is a comment?
  - Single line comment
  - Multiline comment
- Code standards & conventions
  - Naming conventions
  - Indentation
- Comparison Operators
- Making decisions
- Logical operators