Python 1

Overview

- Basics of Computing
- The Python Programming Language
- Data Types
- Making Decisions
- Inputs and Outputs
- Logical Operators
- Questioning

Learning Objectives

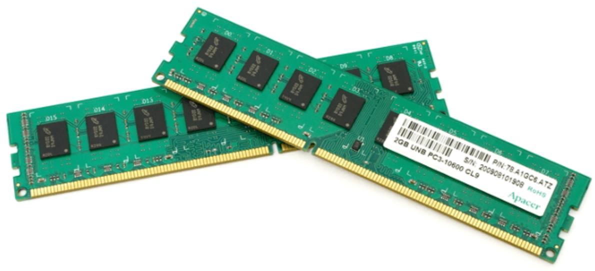
- Describe how a computer is composed by its varying hardware.
- Know the difference between interactive mode and script mode in Python.
- Explain what a variable is.
- Identify the differences between logical operators and comparison operators.
- Write programs in Python using different data types, as well as input and output operations.
- Identify what makes a good question when asking for help

Basics of Computing

What is a computer?

- A computer is a device which processes information according to a set of given instructions
- Composed of various pieces of hardware

Can you name these hardware components?



Are any of these computers?

What about these?

NetBSD
>>> Toaster ready to accept toast.
Press 'Cancel' button to abort
Total toasted: 159, Burnlevel:
Current CPU board temp is

What is data?

- Units of information, usually numerical, that is collected through observation
- A set of values of qualitative or quantitative variables about one or more persons or objects
- Used throughout humanity for science, business, finance, governance and more

---

What is computer data?

- Data that is represented using the binary number system of ones (1) and zeros (0), as opposed to analog representation
- How we use this data is what this course is all about

A brief explanation of binary

https://www.youtube.com/watch?v=Xpk67YzOn5w

Quiz Time! 🤓

What is the decimal representation for the binary value `1011`?

1. 11
2. 13
3. 22
4. 26

Answer - `1: 1 + 2 + 0 + 8 = 11`

What is to program?

- Computers are dumb, they need to be told explicitly how to do something, each and every time.
- Programming is the act of instructing the computer how to perform a specific task as a sequence of logical steps in a language it understands
- How do we program? Well...

Python

Preface

- Like learning any new skill, programming **will be difficult** at first. Stick with it and you will see results!
- This course will **not** turn you into a good programmer overnight, but will get you on the right track.
- Practice, practice, practice.

Anything worth doing is difficult at first!

A Brief History

- Created by Guido van Rossum in 1991 (In software terms, that's a long time ago!)
- Free and open source
- Can run on (probably) any operating system
- The latest main version (3.x) was released in 2008
- Python 2 was discontinued in 2020 so make sure to use Python 3

Always use Python 3

You can check your Python version by typing the following into a terminal:

```
# Windows
$ py --version

# MacOS/Unix
$ python3 --version
```

Note: Don't include the $ sign when copying commands from the slides. The $ sign indicates that it is the command prompt, and we should copy the text directly after it.

Running Python Code

There are two ways to run Python code:

- Interactive mode
- Script mode

Interactive Mode

- Provides a quick way of running lines of code
- Useful when you want to execute basic commands

To run in interactive mode, run the Python command:

```
# Windows
$ py
```

```
# MacOS / Unix
$ python3
```

You should see something like this:

```
Python 3.8.4 (v3.8.4:dfa645a65e, Jul 13 2020, 10:45:06)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Interactive Mode - Example

Type this line of code and hit enter:

```
1 + 2
```

What happened?

Exiting Interactive Mode

You can run the command `exit()` to exit the interpreter.

```
>>> exit() # currently inside the interpreter
$          # back to the original terminal
```

Interactive Mode - Pros and Cons

Pros:

1. Good for beginners.
2. Helpful when we only have a few commands we want to run.

Cons:

1. You can't save the code you've written for long-term use.
2. Tedious to run larger amounts of code.
3. Not a good strategy for running code in the long run.

Script Mode

- Most of the time, code will be stored in script files.
- Script files are ran top-to-bottom by Python.

- Python filenames always ends in `.py`.

Script Mode - Example

Open `VSCode`, create a new file called `hello.py` and type the following, then save it:

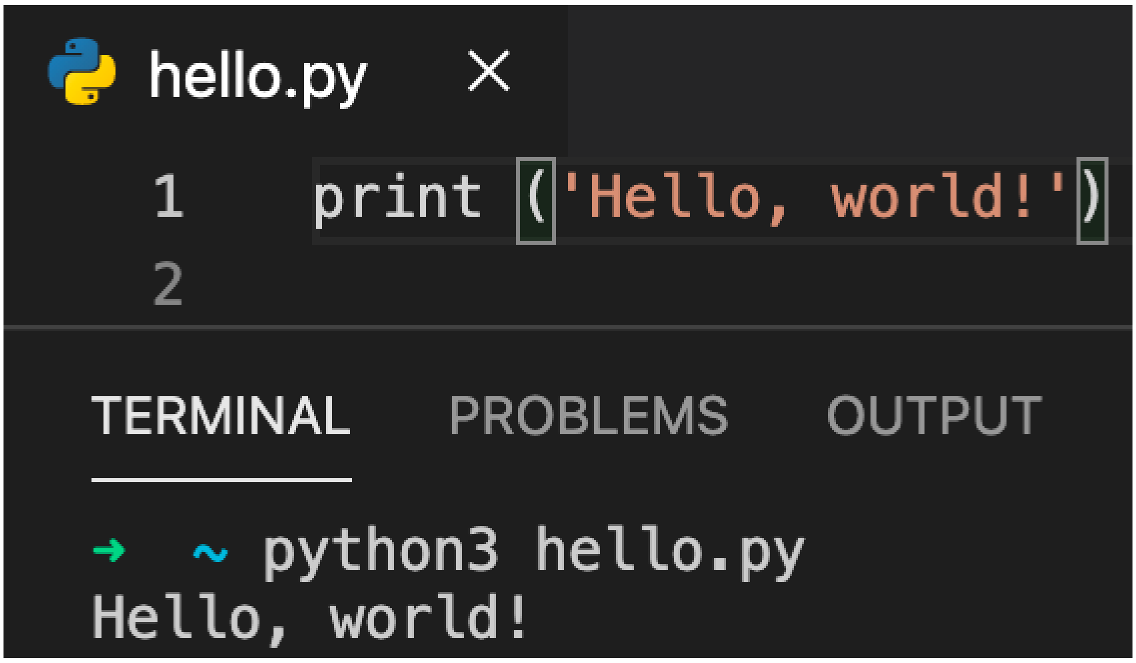
```
# outputs 'Hello, World!' to the terminal
print ('Hello, world!')
```

Run the program on the terminal:

```
$ py hello.py
```

What happened?

Script Mode - Output



```
print ('Hello, world!')
```

```
→  ~ python3 hello.py
Hello, world!
```

Numbers and Maths

- Programming languages are just like calculators, only more complex.
- You can perform calculations with mathematical symbols and numbers.
- Not all symbols match up to real life mathematics.

Arithmetic Operators

- Calculations can be performed in Python.

- The output will be the result of the operation.

```
Operator    Name        Example

+           plus        x + y

-           minus       x - y

/           divide      x / y

*           multiply    x * y

%           modulus     x % y

()          brackets    (x + y) * (y - x)
```

Remember that order of operations still apply in Python! (PEMDAS)

```
>>> (5 * 10) - (20 + 10)
20
>>> 5 * 10 - 20 + 10
40
```

Comparison Operators

- Comparison operators are used to compare two values.
- The output will either be true or false.

```
Operator    Name                    Example

==          is equal to             x == y

!=          is not equal to         x != y

>           greater than            x > y

<           less than               x < y

>=          greater than or equals  x >= y

<=          less than or equals     x <= y
```

```
>>> 10 == 10
True
```

```
>>> 10 == 11
False
```

Quiz Time! 🤓

Q: What is the correct output for this Python command? Try and work it out in your head.

```
>>> 10 + 20 / 4 * 10
```

1. `0.75`
2. `60`
3. `True`
4. `60.0`

Answer: `4`

We will come onto this later...

Variables

As of now we have only been dealing with *constant* values, such as *20*.

We often want to **store** values somewhere when writing a program.

How do we do this? Why are they needed?

What is a variable?

- A way of **storing** data in a program, such as text, numbers and much more.
- Values are stored in computer memory (RAM).
- Variables always have names, to describe what they are storing.

Why do we need variables?

- Allows us to keep track of data we are using in the program.
- Clearer and easier to understand code with labels.

How do I use a variable?

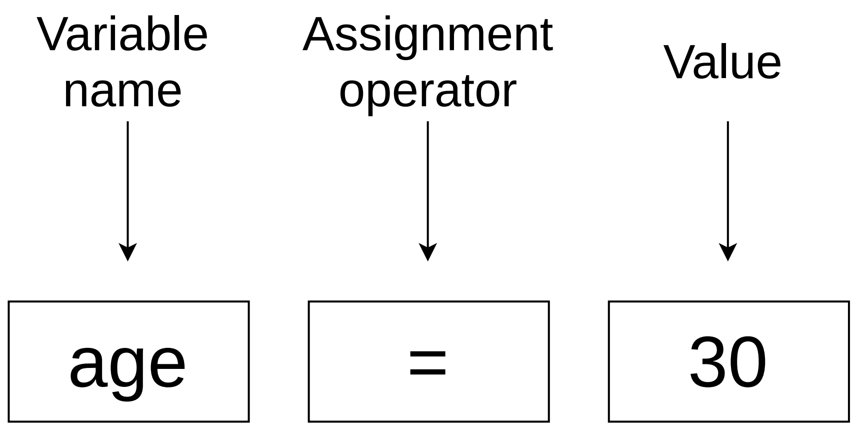
We use variables in real life. Here are some examples:

- My *age* is *30*
- My *name* is *John*
- I am an *adult*. This is *true*.

In Python, this would look like this:

```
age = 30
name = 'John'
is_adult = True
```

Variable Anatomy



Variable name - The label of the data we wish to store.

Assignment operator - How we assign data to the variable.

Value - The data we are storing.

Watch out for = and ==

Always remember that:

- `=` **assigns** a value to a variable.
- `==` **compares** data for equality.

What would happen in the below examples?

```
one = 1
two = 2
one = two
print(one)
```

```
one = 1
two = 2
one == two
print(one)
```

Modifying Variable Data

You can **change** the data stored in variables.

Examples:

```
age = 30 # initial value for variable
print(age)
>>> 30

age = 40 # value has now changed
print(age)
>>> 40
```

Modifying variable data

You can also assign variables to variables.

This works by assigning the value stored in one variable, to another variable. Example:

```
a = 1
b = 2

a = b
print(a)
```

What will be printed for variable `a`?

Naming Variables

All of the below are valid ways to name a variable

```
my_age = 21
myAge = 21
MyAge = 21
MYAGE = 21
```

You should use the first example: `my_age = 21`.

This makes your code easier to read. For more information on naming variables, see here.

We will see other concepts later in the course that use different naming standards.

Quiz Time! 🤓

What will the value of these three variables be?

```
a = 1
b = 2
c = 3

a = c
c = b
b = a
a = c
```

1. `a = 3, b = 2, c = 1`
2. `a = 1, b = 2, c = 3`
3. `a = 2, b = 3, c = 2`
4. `a = 3, b = 2, c = 3`

Answer: `3`

Data Types

Data Types

Variables can store data of different types:

Basic Types:

```
a = 100 # Integer (int)

b = 2.5 # Decimal (float)

name = 'Bob' # String (str)

is_active = True # Boolean (bool)
```

Data Types

Additional Types:

```
position = (12.33, -122.34) # Tuple

colours = ["Cyan", "Magenta", "Yellow", "Black"] # List

numbers = {1, 2, 3, 4, 5, 6} # Set

# Dictionary (dict)
data = {
  "key": "value",
  "another_key": False,
  "some_items": [1, 2, 3]
}
```

We will explore all these individually later on.

Static Types

In some languages, you specify the type of the variable.

This is called a ***static type***.

```
// java
int age = 30
String name = "John"
Boolean isHappy = true
```

Dynamic Types

In Python, you don't need to specify the type, as it will work it out for you.

This is called a ***dynamic type***.

```
age = 30
name = 'John'
isHappy = True
```

Weak Typing

In some languages, you can perform operations with different types of data.

This is called ***weak typing***.

```
// javascript
my_variable = 1 + 'hello';

// output: "1hello"
```

Strong Typing

In Python, you are not allowed to do anything that's incompatible with the type of data you're working with. For instance, you can't add a string and integer together.

This is called **strong typing**.

```
# Throws an error
my_variable = 1 + 'hello'

# TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Quiz Time! 🤓

Q: Python is...

1. Strongly and statically typed
2. Strongly and dynamically typed
3. Weakly and statically typed
4. Weakly and dynamically typed

Answer: 2

Working with Data

Expressions

- An expression is something which can be evaluated to produce a result
- It's made up of operands and operators
  - Operands are values
  - Operators are arithmetic symbols (+ - * /)

Examples:

```
4 + 7 / 2

my_first_value - my_second_value
```

```
'Hello' + ' World'
```

Comments

You may have noticed something in a previous slide:

```
# Throws an error
my_variable = 1 + 'hello'
```

- The wording after the hash symbol is called a comment
- Comments can be used to help others (and yourself) understand the code you've written better
- Python will ignore comments and thus not run as executable code

---

Strings

Strings in Python are just **text**.

Python strings are wrapped in single (`'`) or double (`"`) quotes:

```
# Basic String
name = 'John'
```

A string can also span multiple lines if needed with three single (`'`) or double (`"`) quotes:

```
# Multi-Line String
main_menu =
'''
    Please choose from the following:

    [1]: View Balance
    [2]: View Account Details
'''
```

String Manipulation

You can combine strings together:

```
a = 'Hello,'
b = ' world!'
print(a + b)
```

```
>>> Hello, world!
```

This is known as **concatenation**.

String Interpolation

- Allows us to put variables inside strings. The variable names will be replaced with their value at runtime.
- Prepend an `f` to the beginning of a string to use interpolation.

Example:

```
amount = 12.40
merchant = 'Amazon'
transaction_str = f'You spent £{amount} at {merchant}'
```

```
>>> You spent £12.40 at Amazon
```

Try it out!

Lists

Lists hold zero or more elements of *indexed* data. This data can be of any type.

```
empty_list = []

numbers = [1, 2, 3, 4, 5]

people = ["John", "Jane", "Janet"]

mixed = ["John", 2, "Jane", 3, False]
```

List Indexing

Most programming languages start their indexing from 0 instead of 1, this is important to remember!

That is, the first element in a list starts at position 0.

```
people = ["John", "Jane", "Janet"]

people[0] # John
```

```
people[1] # Jane
people[2] # Janet
```

List Selection

You can select parts of a list by using the following syntax:

```
people = ["John", "Sally", "Mark", "Lisa"]

# Get from index 1 up to but not including index 3.
people[1:3]
>>> ['Sally', 'Mark']

# Get last item
people[-1]
>>> 'Lisa'

# Omitting left-hand side means "start from the beginning"
people[:-1]
>>> ['John', 'Sally', 'Mark']

# Get all items but the first
# Omitting right-hand side means "end of the list"
people[1:]
>>> ['Sally', 'Mark', 'Lisa']
```

Inputs and Outputs

--

Standard Output

How can we display information to a user?

Python has an in-built `function` for outputting data:

```
# Printing text
print('Hello World')

# Printing a variable
message = 'This is a message'
print(message)

#Printing a list
people = ['John', 'Sally', 'Mark', 'Lisa']
print(people)
```

Standard Input

How can we collect data from a user?

Python has an in-built `function` for inputting data:

```
#input with no prompt
text = input()
print("You entered:" + text)

#input with a prompt
name = input("What is your name? ")
print("Nice to meet you, " + name)
```

Converting Input Values

`input()` will always return a string. What if we want the value to be a different type, such as integer or float?

If we want to convert it to an integer for instance, we can wrap `input()` in `int()`:

```
num = int(input('Enter a number: '))
print(f'This square of this value is {num * num}')
```

```
Enter a number: 5
This square of this value is 25
```

Quiz Time! 🤓

Q: Consider the Python list, what values will be printed?

```
values = ['A', 'B', 'C', 'D', 'E', 'F']
print(values[1:4])
```

1. `A, B, C, D`
2. `B, C, D`
3. `B, C, D, E`
4. `All of them`

Answer: `2`

Exercise

Please refer to **Part 1** of the exercise handout.

Making Decisions

Making Decisions

In programming, often we want to make certain decisions based on a **condition**.

Example: Suppose you are writing a game program. You may want to build these features:

- **IF** the user presses the up arrow, **THEN** the character will move forward.
- **IF** the user presses spacebar, **THEN** the character will jump.

So how do we do this in Python?

Conditionals

We looked at comparison operators before. They are also known as conditionals.

Conditionals are expressions which return **True** or **False**.

These will help us make *decisions* in our programs.

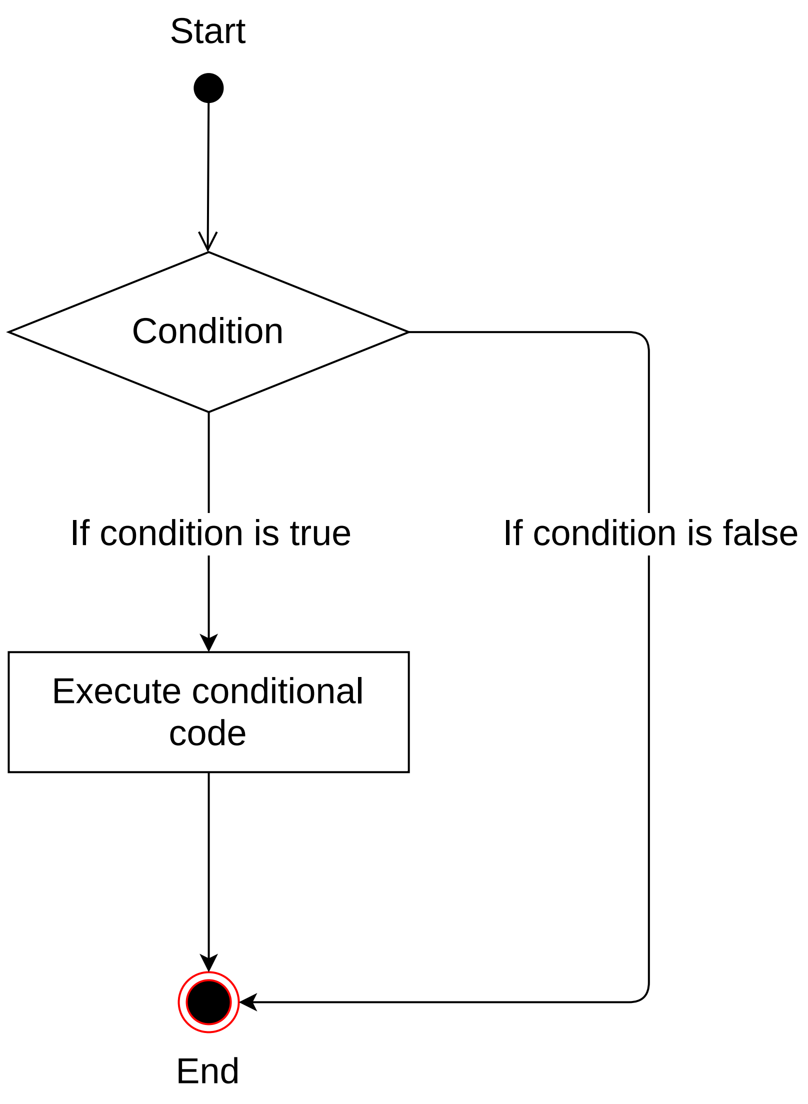
```
'A' == 'B'   # False

100 > 5      # True

9 < 2        # False

a = 1
b = 2
a < b        # True
```

Anatomy of a Decision

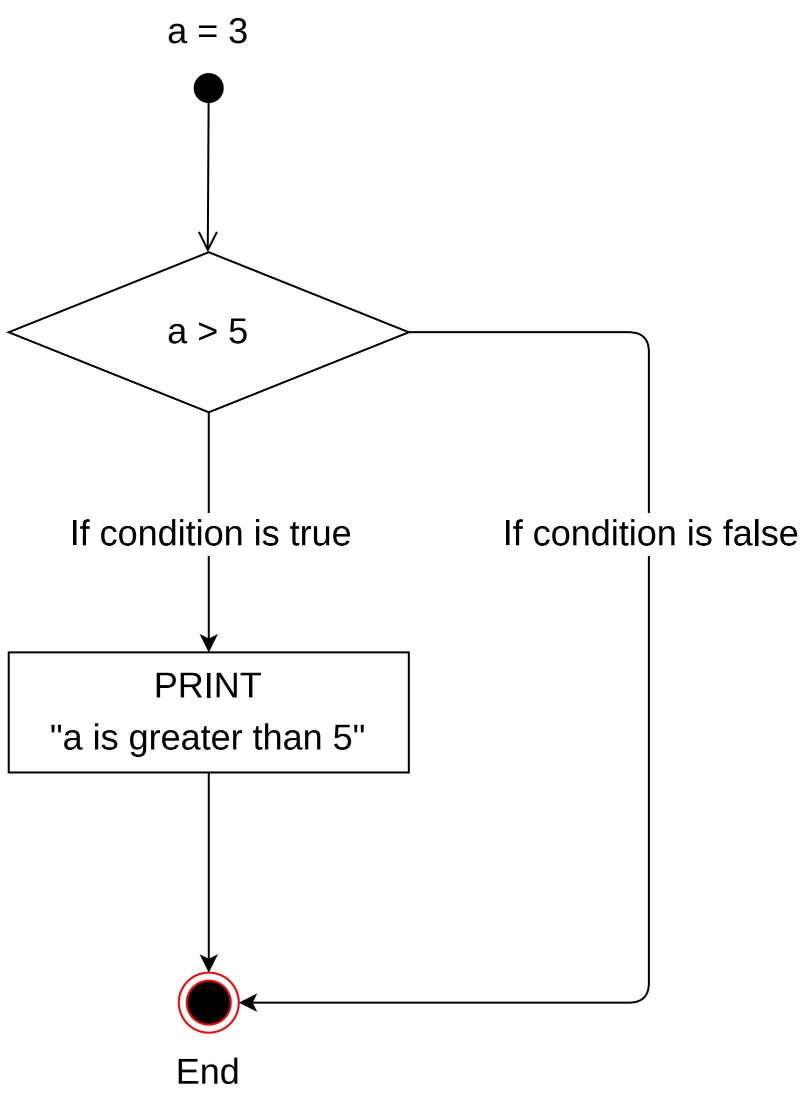


Control Flow: The order in which individual statements are executed

---

Example Decision

- Imagine we have given the value of 3 to `a`.
- If `a` is greater than 5, we will print that it is so.
- If `a` is less than 5, we will do nothing.



Decision Making in Python

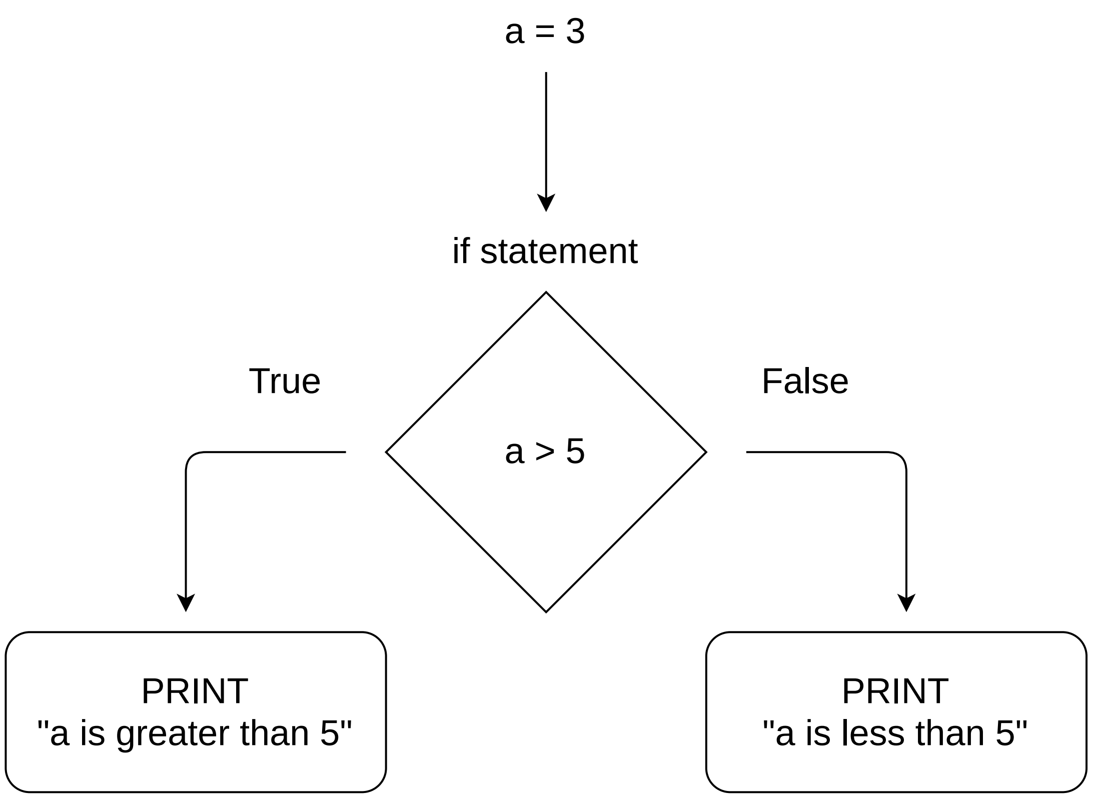
Python uses `if` statements, which cause the program to branch out to a different part of the code, depending on some **condition**.

```python
a = 3

if a > 5:
    print("a is greater than 5")
```

If the condition is met, the indented statement(s) directly below it will execute.

What about when the condition is false?



If / Else

In the previous example, `a` was less than `3`. What if we want to do something when a condition is false?

Python has this functionality, with the `else` statement.

```
a = 3

if a > 5:
    print('a is greater than 5')
else:
    print('a is less than 5')
```

Note: You cannot use `else` without `if`, but you *can* use `if` without `else`.

If / Else If

You can evaluate multiple conditions with `if` / `else if`.

The `else if` keyword in Python is `elif`.

```python
a = 12

if a > 15:
    print('a is greater than 15')

elif a > 10:
    print('a is greater than 10')

elif a > 5:
    print('a is greater than 5')

elif a > 0:
    print('a is greater than 0')
```

What will be printed?

Multiple If Statements

You can use multiple `if` statements one after the other.

The difference this time is that each `if` statement is evaluated, regardless of if the previous evaluated to true.

```python
a = 12

if a > 15:
    print('a is greater than 15')

if a > 10:
    print('a is greater than 10')

if a > 5:
    print('a is greater than 5')

if a > 0:
    print('a is greater than 0')
```

What will be printed?

Nested If Statements

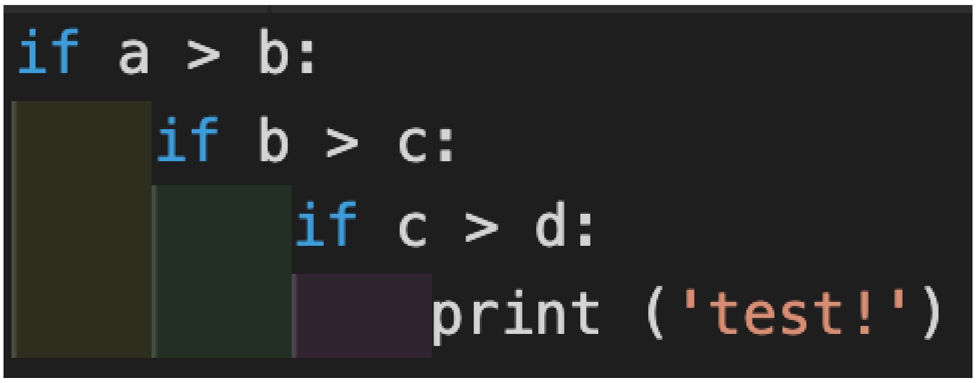
You can *nest* `if` statements inside `if` statements:

```python
a = 10
b = 20
```

```
if a >= 10:
    if b >= 10:
        print ('a and b are greater than or equal to 10')
```

Indentation Matters

- Indentation denotes the structure of your program
- If you've seen code in a "curly brace" language before, indentation is the Python equivalent of that
- Use the **tab button** to insert an indent



```
if a > b:
    if b > c:
        if c > d:
            print ('test!')
```

Exercise

Please refer to **Part 2** of the exercise handout.

Logical Operators

Logical Operators

Logical operators are used to combine conditional statements:

```
Operator    Description                             Example

and         true if both statements are true        x > 5 and x < 10

or          true if at least one statement is true  x > 5 or x < 10

not         reverse the result                      not x
```

```
# true if both statements are true
x > 5 and x < 10

# true if either or both statements are true
x > 5 or x < 10
```

```
# reverse of the result
not x
```

Understanding How it Works

Logical operators become simple to understand once you see the maths behind it.

The next slide will demonstrate this using something called **binary truth tables**.

Truth tables can be used to describe logical expressions.

Binary Truth Tables - AND

Imagine we have two conditions, `a` and `b`:

```
# a          b
x > 5 and x < 10
```

a	b	AND
T	T	T
T	F	F
F	T	F
F	F	F

For this table, `T` is `true` and `F` is `false`.

Binary Truth Tables - OR

```
# a         b
x > 5 or y < 10
```

a	b	OR
T	T	T
T	F	T
F	T	T
F	F	F

Binary Truth Tables - NOT

```
# a
not x
```

a	NOT
T	F
F	T

Example 1

You can integrate logical operators with `if` statements:

```
age = 17
filmRating = 15

if age < 18 and filmRating == 18:
  print("You're too young to watch this film")

else:
  print("You're old enough to watch this film")
```

Example 2

```
age = 17
filmRating = 15

if  age < 12 and filmRating == 12 or
    age < 15 and filmRating == 15 or
    age < 18 and filmRating == 18:
    print("You're too young to watch this film")

else:
  print("You're old enough to watch this film")
```

Quiz Time! 🤓

Q: Consider the following code snippet, what will the output be?

```
x = -15
y = 10

if x > 0:
  if y > 0:
    print('both x and y are positive')      # 1
  else:
    print('x is positive, y is negative')   # 2

else:
  if y > 0:
    print('x is negative, y is positive')   # 3
  else:
    print('both x and y are negative')      # 4
```

Answer: `3`

Exercise

Please refer to **Part 3** of the exercise handout.

Asking questions

Good and bad questioning practices

Some bad examples:

what does this error mean? `<20 lines of pasted error message>`

`<long pasted error message with no context>`

Hey my git isn't working, what should I do?

And some good ones:

I'm trying to run this merge statement into a delta table however I'm getting this error

```
"pyspark.sql.utils.ParseException: mismatched input 'MERGE'"
```

```
MERGE INTO {table_name} AS stg
 USING tmp_cohort AS tmp
 ON stg.UNIQUE_REFERENCE = tmp.UNIQUE_REFERENCE and
stg.FILE_NAME=tmp.FILE_NAME
 WHEN MATCHED THEN UPDATE SET *
 WHEN NOT MATCHED THEN INSERT *
```

When I run this insert everything is fine

```
INSERT INTO {table_name}
SELECT * FROM tmp_cohort
```

so the schemas are correct. I've been looking around online and was under the impression merge's may not be supported?

Another good one:

I'm trying to run the following script: grant_user_permissions.py but whenever I run it logged in as the base user I get permission denied. I don't have admin permissions, but looking at the script I don't think I need them. Is there something I'm missing here?

Some tips for good questioning

1. Be as specific you can be about what is going wrong and the environment it is going wrong in
2. Before asking the question, think about what you want out of the answer and be explicit about it
3. Make it clear that you have attempted to solve it, and the understanding that you currently have
4. Read your message before sending it, and identify what more information you would want if it was sent to you - then add it!

Learning Objectives Revisited

- Describe how a computer is composed by its varying hardware.

- Know the difference between interactive mode and script mode in Python.
- Explain what a variable is.
- Identify the differences between logical operators and comparison operators.
- Write programs in Python using different data types, as well as input and output operations.
- Identify what makes a good question when asking for help

Terms and Definitions Recap

Hard Drive: For storing all of the data on our machine for long-term use

CPU: Used for executing instructions that make up a computer program

RAM: A form of computer memory used for accessing data in a faster way than a hard drive

Binary: A number expressed in the base-2 numeral system, which only uses 0 and 1

Terms and Definitions Recap

Static Typing: Variables of a language do not need their type to be defined before they're used

Dynamic Typing: Variables of a language must have their type defined before they are used

Weak Typing: A value has a type but has the ability of changing

Strong Typing: A value has a type and cannot be changed

Terms and Definitions Recap

Interpreter (Interactive Mode): A program that directly executes instructions written in a programming language without requirement of compilation

Control Flow: The order in which individual statements are executed

---

Further Reading

- The Hitchhiker's Guide to Python
- Python Documentation
- Why Python is dynamic and also strongly typed