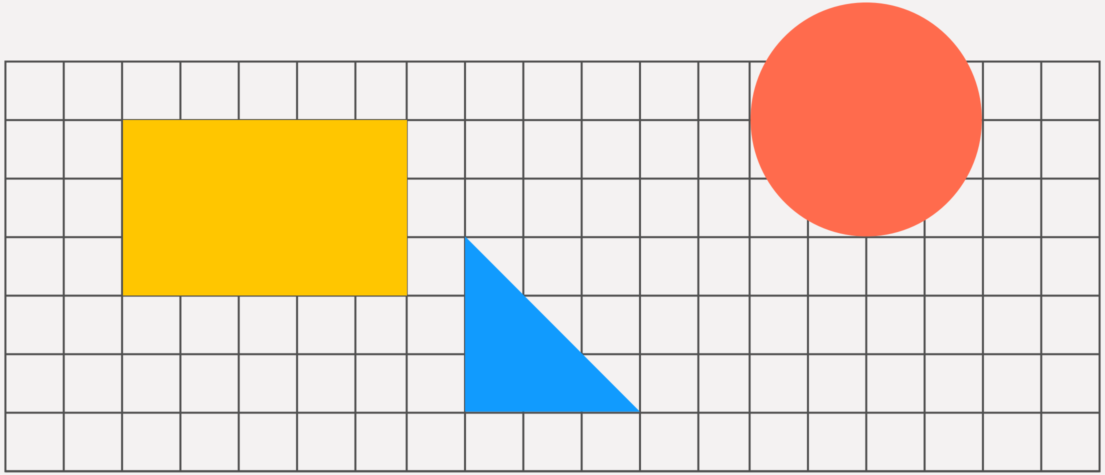
The Solid Snake!

Ali Abrishami



In This Lecture You Will...

a

Learn what Python is

b

Discuss its pros and cons

c

Learn different programming paradigms

Here Comes Python!

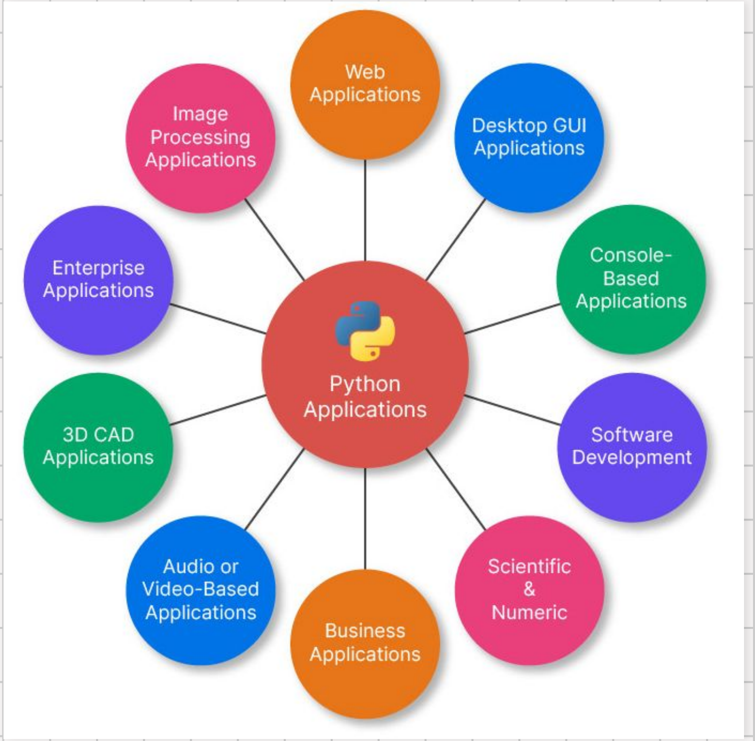
- Python was created by Guido van Rossum.
- First release on February 20, 1991.
- The name of the Python programming language comes from an old television comedy sketch series called Monty Python's Flying Circus.

Joys of Learning Python

- It is easy to learn.
- It is easy to use for writing new software.
- It is easy to obtain, install and deploy.
- You can tell your Potterhead friends you know Parseltongue :)

Python Applications

- Web development
- Data science
- Artificial intelligence
- Automation
- Scripting



Your Knowledge Portfolio

- An **investment** in **knowledge** always pays the **best** interest.
- Your knowledge and experience are your most important professional assets.
- Unfortunately, they're expiring assets.
- A smart investor practices:
  - Serious investors invest **regularly**—as a habit.
  - **Diversification** is the key to long-term success.
  - Smart investors balance their portfolios between conservative and high-risk, **high-reward** investments.
  - Investors try to buy low and sell high for maximum return.
  - Portfolios should be reviewed and rebalanced periodically.

Your Knowledge Portfolio

- Learn at least one new language every year.
- By learning several different approaches, you can help broaden your thinking and avoid getting stuck in a rut.
- Read a technical book each quarter. Once you're in the habit, read a book a month. Read nontechnical books, too.
- Take classes.
- Participate in local user groups.
- The process of learning will expand your thinking, opening you to new possibilities and new ways of doing things.

Python Key Features

- Cross-platform Language
- High-Level Language
- Easy to learn and use
- Dynamic Language
- Large community and extensive libraries
- Object-Oriented Programming Language
- Free and Open-source libraries

```
public class Main {
public static void main(String[] args) {
System.out.println("Hello, World!");
}
}
```

```
#include <iostream>
using namespace std;
int main() {
cout << "Hello, World!" << endl;
return 0;
}
```

```
#include <stdio.h>
int main() {
printf("Hello, World!\n");
return 0;
}
```

```
print("Hello, World!")
```

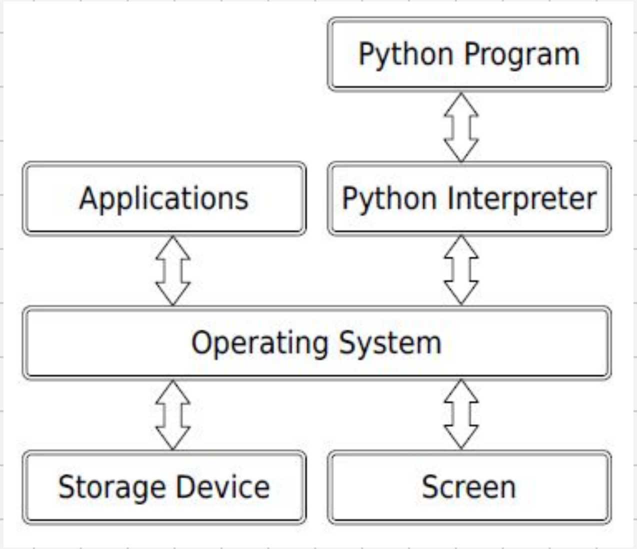
Java
imgflip.com

Open Source World!

- **Cost-Efficiency**: No licensing fees; it's free to use.
- **Transparency**: You can see the code and understand how it works.
- **Community Support**: Large communities contribute to and maintain the code, which means robust documentation and support.
- **Flexibility**: You can modify the code to suit your needs.
- **Security**: Many eyes on the code means bugs and security issues are identified and fixed quickly.
- **Innovation**: Fast-paced development with contributions from developers around the globe.

It's Interpreted?

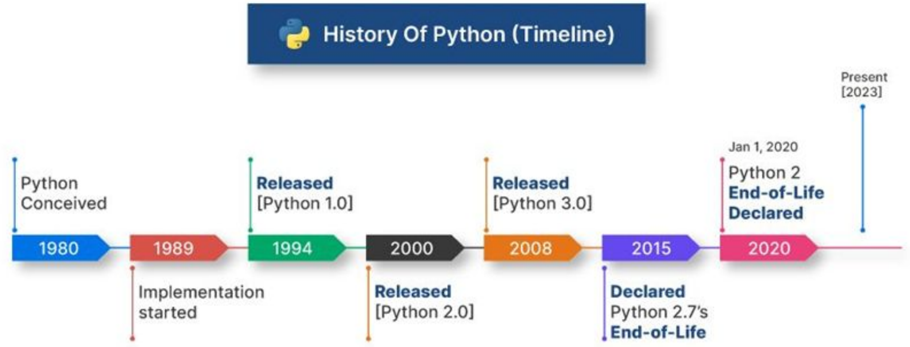
- When you write a program in Python, Java, or …, it doesn't run directly on top of the operating system.
- Instead, another program, called an interpreter or virtual machine, takes your program and runs it for you.
- Translating your commands into a language the OS understand.



Notice the hierarchy here!

Python versions

- Python reached version 1.0 in January 1994.
- Python 2.0, released October 2000.
- Python 3.0, released December 2008.
- Python 3.14, released October 2025.



Install Python

- Open a browser window and navigate to https://www.python.org/downloads/
- Under the “Download the latest version for Windows” heading, click the download button.
- After completion of .exe file download, run the installer.
- Verifying installation with: `python --version` or `python -V` in command line.
- Then you can either use a shell or your IDE of choice to operate it!

VirtualEnv

- In Python, virtualenv is a tool used to create isolated environments for your projects.
- Each project can have its own dependencies, packages, and Python version, independent of your system’s global Python installation or other projects.
- Steps:
  - pip install virtualenv
  - virtualenv -p python3.14 myenv
  - myenv\Scripts\activate (on Windows) *or* source myenv/bin/activate (on Linux)

Data Types in Python

- `int`: For storing integer numbers.
- `float`: For storing floating point numbers.
- `str`: For storing strings.
- `bool`: For storing booleans.

Class Activity Time!

What does `17.0/10.0` produce in Java? What about Python?

What does `17/10.0` produce in Java? What about Python?

What does `17.0/10` produce in Java? What about Python?

What does `17/10` produce in Java? What about Python?

Finite Precision Problem

- Floating-point numbers are not exactly the fractions you learned in grade school.
- The problem is that computers have a **finite amount of memory**.
- Most programming languages limit how much information can be stored for any single number.
- The number `0.6666666666666666` turns out to be the closest value to `2/3`.

Arithmetic Operators

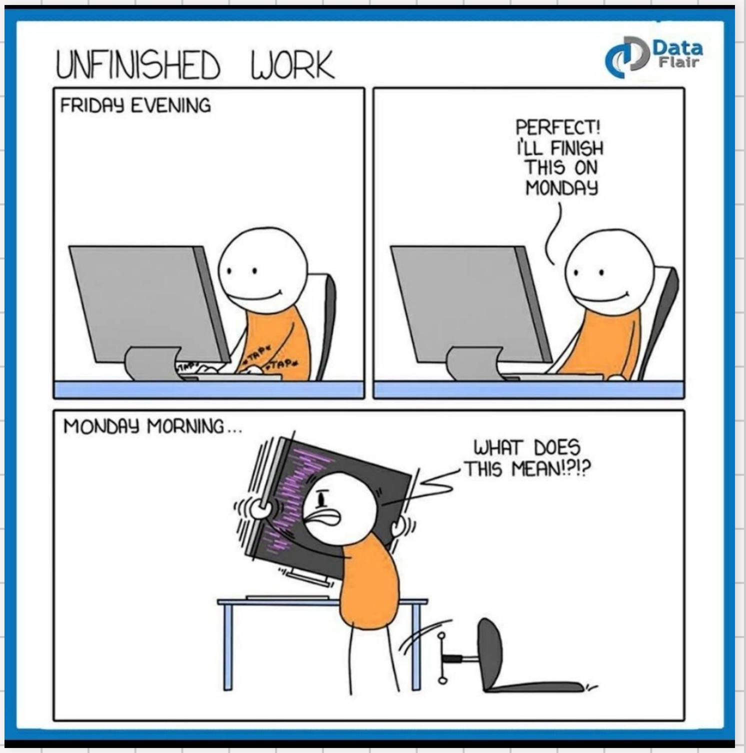
- `12+10`: Addition, results in 22.
- `12-10`: Subtraction, results in 2.
- `12*10`: Multiplication, results in 120.
- `12/10`: Division, results in 1.2.
- `12//10`: Integer division, results in 1.
- `12%10`: Modulo, results in 2.
- `12**10`: Power, results in 61917364224.

Variables

- Variables can be declared just like they do in C and Java.
- The key difference is you don’t need to specify the type (although you can, in a way).
- When unspecified, variables can change types!
- The laws of variable naming in languages like C and Java also apply here!
- Example: `darth_vader = “cool!”`

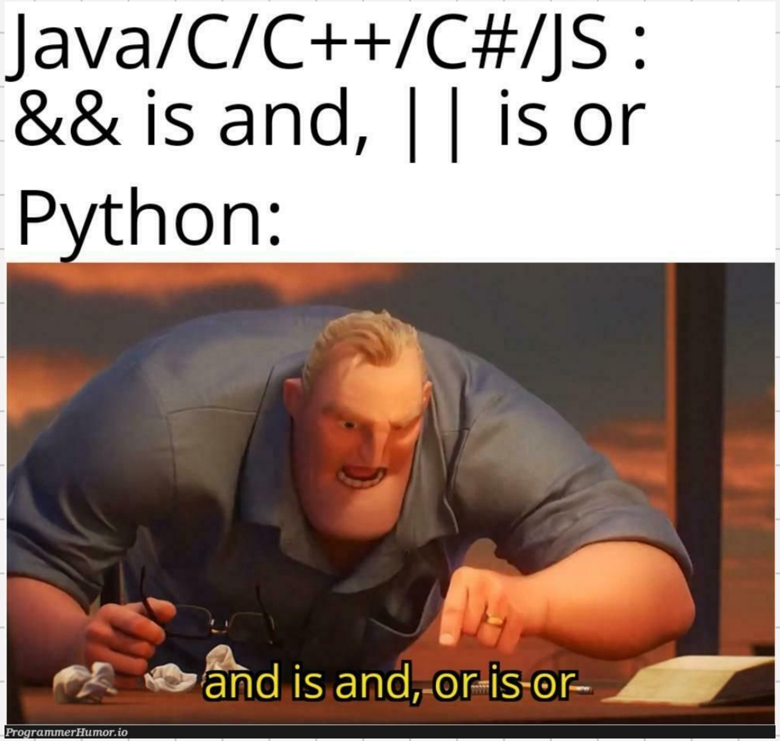
Comments

- Just like in C and Java, you can clear things up by writing comments.
- This will clear things up, for your own future self and your poor coworker!
- In Python, any time the # character is encountered, Python will ignore the rest of the line.
- Example: `variable1 = 2 # this is some random variable`



Boolean Operators

- There are only three basic Boolean operators:
  - and
  - or
  - not
- Order of precedence (highest to lowest):
  - not
  - and
  - or



Relational Operations

- `==`: Equal to
- `!=`: Not equal to
- `>`: Greater than
- `<`: Less than
- `>=`: Greater than or equal to
- `<=`: Less than or equal to

False Values

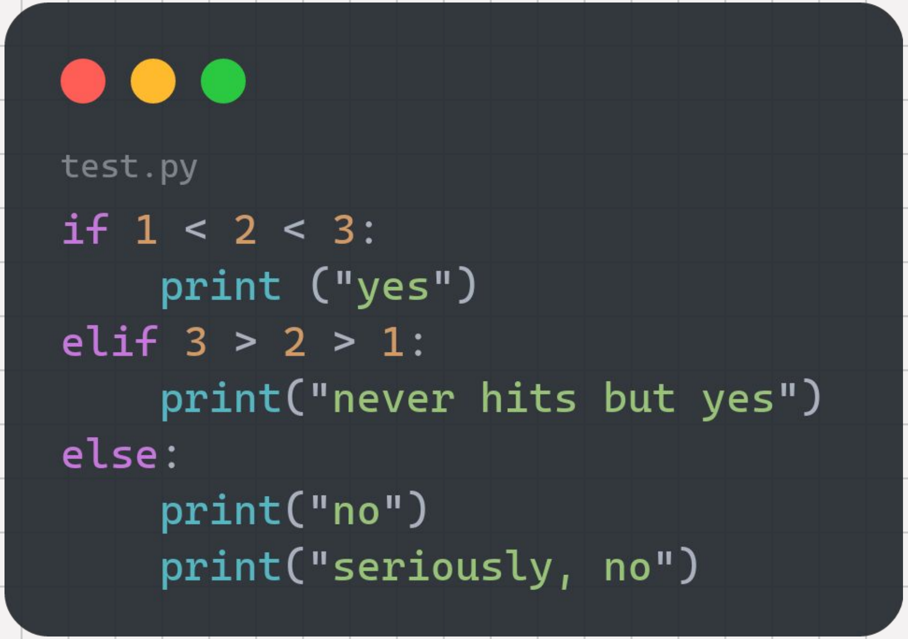
- `None`: The none value is evaluated as False.
- `False`: The literal boolean value that already means “no.” (opposite of `True`)
- `0`: A number with zero magnitude, treated as empty in logic checks.
- `0.0`: Same emptiness as 0, just in floating-point form.
- `""`: An empty string with no characters, read as “nothing to say.”

Short Circuit!

- `and` stops on `False`: It halts the moment it finds a value that can't make the whole expression `True`.
- or stops on `True`: It returns the first value that can make the whole expression `True`.
- Returns actual values: Short-circuiting gives back the value it stopped on, not a strict boolean.
- Skips unnecessary work: Later expressions don't run if the answer is already known.
- Protects from errors: Lets you safely check conditions like `x` and `x.value` without crashing.

Conditional Statements

- We can use an if statement to execute a block only when the statement is true.
- If the condition is false, the statements in the block aren't executed
- If none of the conditions in a chain of if/elif statements are satisfied, Python does not execute any of the associated blocks. We may as well add an `else` too.



test.py

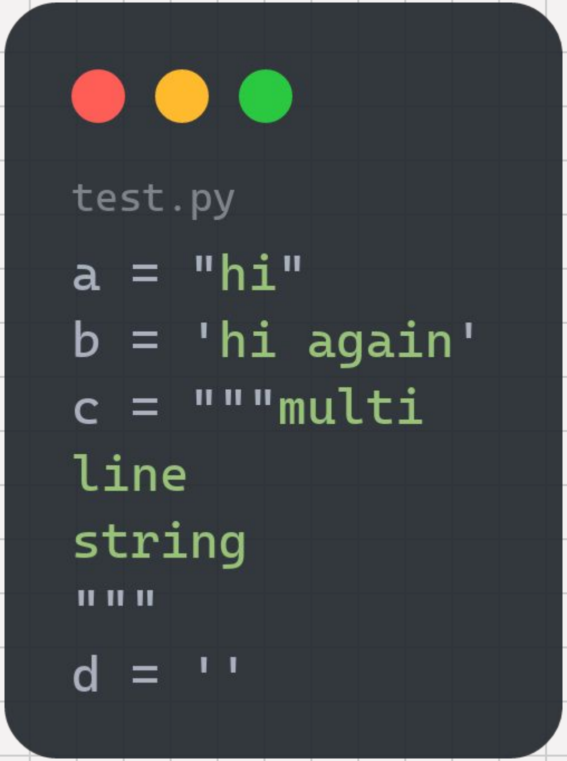
```python
if 1 < 2 < 3:
    print ("yes")
elif 3 > 2 > 1:
    print("never hits but yes")
else:
    print("no")
    print("seriously, no")
```

Ternary Operator

- Ternary operator is for writing simple if-else conditions in a single line:
- **true_value if** *«condition»* **else false_value**
- result = “Positive” if x > 0 else “Non-Positive”

Strings

- Just like strings in Java.
- Can be concatenated using `+`
- You can even have multiline strings, just use `’’’` or `”””`.
- Empty strings are the shortest strings!
- You can get the length of a string using `len`. Example: `len(a)` returns 2
- You can also multiply strings! Example: `’A’*2` will be `AA`



test.py

```python
a = "hi"
b = 'hi again'
c = """multi
line
string
"""
d = ''
```

Strings (cont.)

- `ord()` and `chr()`: Convert characters to and from Unicode code points
- `input()` reads input from user
- `'int' in ‘printer’` is `True` because you there is an “int” in “printer”.
- `"Abc123"[2:4]` is `"c1"`, it’s called slicing and it gets

Type Casting

- `int(“12”)`: 12
- `int(12.2)`: 12
- `float(“12.2”)`: 12.2
- `float(12)`: 12.0
- `str(12): ”12”`
- `str(12.2): “12.2”`

Indexing and Slicing

- Strings are sequences, so you can access characters using indices, e.g., `s[0]` for the first character.
- Negative indexing allows access from the end, e.g., `s[-1]` for the last character.
- Slicing extracts substrings, for example, `s[1:4]` gets characters from index 1 up to, but not including, index 4.

String Methods

- `upper()`: Convert to uppercase.
- `lower()`: Convert to lowercase.

- `strip()`: Removes whitespace from both ends.

- `replace(old, new)`: Replaces occurrences of a substring.
- `split(delimiter)`: Splits the string by a delimiter, returning a list.

- `startswith()`: Check if a string starts with a specific substring.
- `endswith()`: Check if a string ends with a specific substring
- `find(substring)`: Returns the lowest index of the substring if found, otherwise returns -1.
- `count(substring)`: Counts the number of non-overlapping occurrences of a substring.

- `isdigit()`: Checks whether all the values in the input string are digits.

String Immutability

- Strings are immutable, meaning once created, they cannot be changed.
- Any operation that modifies a string creates a new one.
- For example, `s.upper()` doesn’t changes but returns a new string in uppercase.
- This design choice has benefits in terms of performance and security, as strings cannot be altered unexpectedly, and memory usage can be optimized for identical strings.

String Formatting

- String formatting in Python is a way of inserting variables or expressions into strings for dynamic text creation.
- 1. f-strings: Simply prefix the string with f and include variables or expressions within curly braces {}. Example: `f"Hello, {name}"`
- 2. format() method: The format() method allows placeholders within curly braces {} that are filled by passing values into the format() function. Example: `"Hello, {}".format(name)`
- 3. Percent (%) Formatting: An older formatting method uses the % operator. Example: `"Hello, %s" % name`

Lists

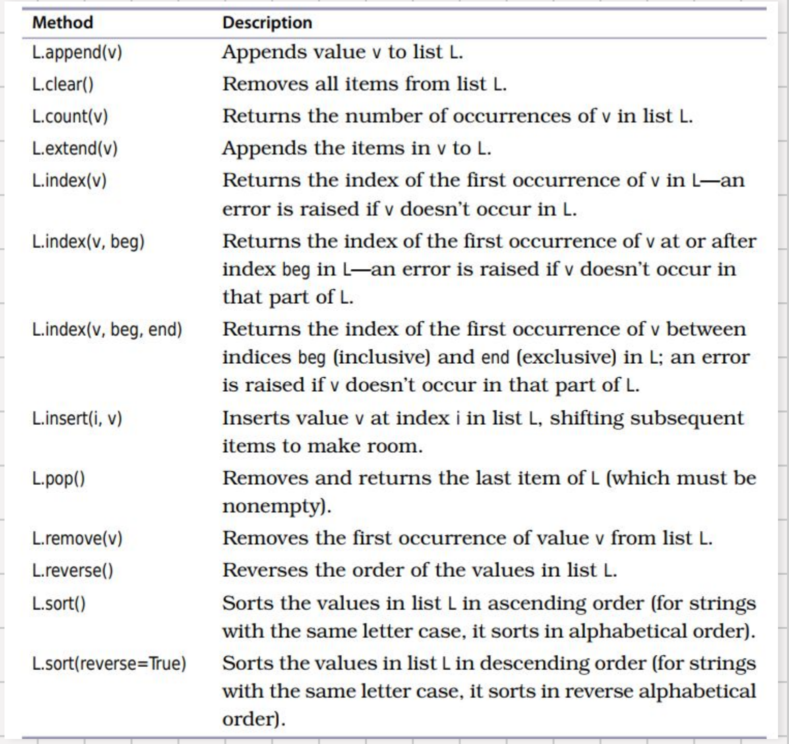
- We can use a list to keep track of multiple objects.
- Indexing and slicing works here; as well as the `len` function and `in` keyword.
- A list is an object; like any other object, it can be assigned to a variable.
- They can be combined using `+` operator.
- Example: `whales = [5, 4, 7, 3, 2, 3, 2, 6, 4, 2, 1, 7, 1, 3]`

Lists Mutability

- List objects are **mutable**.
- That is, the contents of a list can be **mutated**.
- In general, an expression of the form `L[i]` (list `L` at index `i`) behaves just like a simple variable.
- In contrast to lists, numbers and strings are immutable.
  - You cannot, for example, change a letter in a string.
  - Methods that appear to do that, like upper, actually create new strings:

Operations & Methods

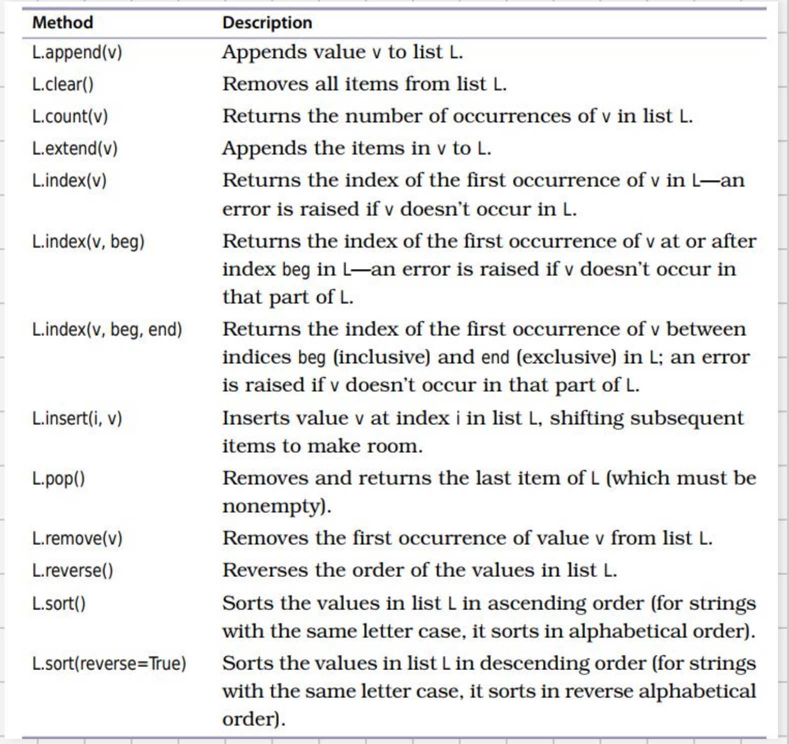
- `len(s)`: count of objects in the list.
- `sum(s)`: sum of numbers in the list.
- `max(s) min(s)`: maximum/minimum number in the list
- `sorted(s)`: returns a sorted copy of list `s`
- `del s[i]`: removes element at position `i`



Method	Description
L.append(v)	Appends value v to list L.
L.clear()	Removes all items from list L.
L.count(v)	Returns the number of occurrences of v in list L.
L.extend(v)	Appends the items in v to L.
L.index(v)	Returns the index of the first occurrence of v in L—an error is raised if v doesn't occur in L.
L.index(v, beg)	Returns the index of the first occurrence of v at or after index beg in L—an error is raised if v doesn't occur in that part of L.
L.index(v, beg, end)	Returns the index of the first occurrence of v between indices beg (inclusive) and end (exclusive) in L; an error is raised if v doesn't occur in that part of L.
L.insert(i, v)	Inserts value v at index i in list L, shifting subsequent items to make room.
L.pop()	Removes and returns the last item of L (which must be nonempty).
L.remove(v)	Removes the first occurrence of value v from list L.
L.reverse()	Reverses the order of the values in list L.
L.sort()	Sorts the values in list L in ascending order (for strings with the same letter case, it sorts in alphabetical order).
L.sort(reverse=True)	Sorts the values in list L in descending order (for strings with the same letter case, it sorts in reverse alphabetical order).

Operations vs. Methods

- Operations on lists are more general actions that can be performed on lists. They don't require dot notation; they use standard Python operators or functions.
- List methods are functions built into the list object in Python, designed specifically to perform actions on lists. They are called directly on a list using dot notation.



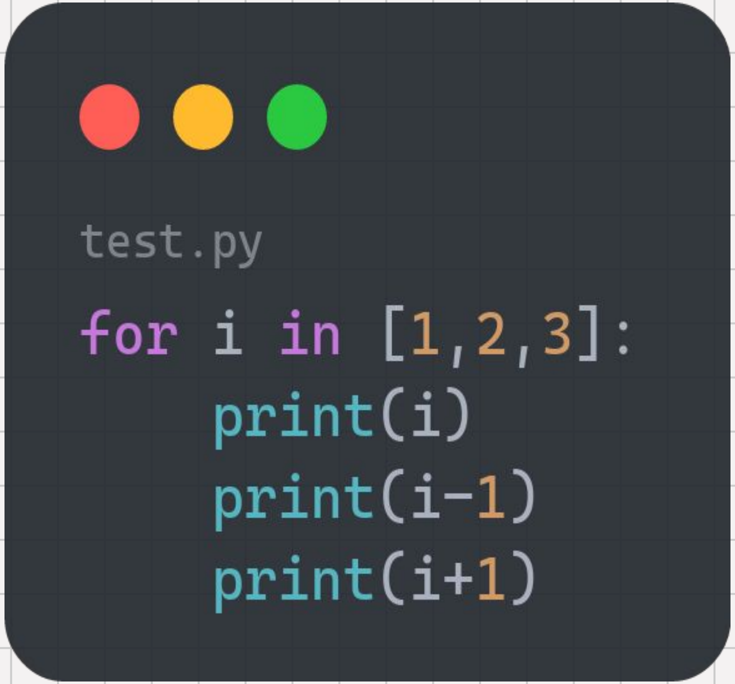
Method	Description
L.append(v)	Appends value v to list L.
L.clear()	Removes all items from list L.
L.count(v)	Returns the number of occurrences of v in list L.
L.extend(v)	Appends the items in v to L.
L.index(v)	Returns the index of the first occurrence of v in L—an error is raised if v doesn't occur in L.
L.index(v, beg)	Returns the index of the first occurrence of v at or after index beg in L—an error is raised if v doesn't occur in that part of L.
L.index(v, beg, end)	Returns the index of the first occurrence of v between indices beg (inclusive) and end (exclusive) in L; an error is raised if v doesn't occur in that part of L.
L.insert(i, v)	Inserts value v at index i in list L, shifting subsequent items to make room.
L.pop()	Removes and returns the last item of L (which must be nonempty).
L.remove(v)	Removes the first occurrence of value v from list L.
L.reverse()	Reverses the order of the values in list L.
L.sort()	Sorts the values in list L in ascending order (for strings with the same letter case, it sorts in alphabetical order).
L.sort(reverse=True)	Sorts the values in list L in descending order (for strings with the same letter case, it sorts in reverse alphabetical order).

For Loops

- The general form of a for loop over a list is as follows.

- The loop variable is assigned the first item in the list, and the loop block is executed. Then The loop variable is then assigned the second item in the list and the loop body is executed again. Finally, the loop variable is assigned the last item of the list and the body is executed one last time.

- Each pass through the block is called an iteration.At the start of each iteration, Python assigns the next item in the list to the loop variable.

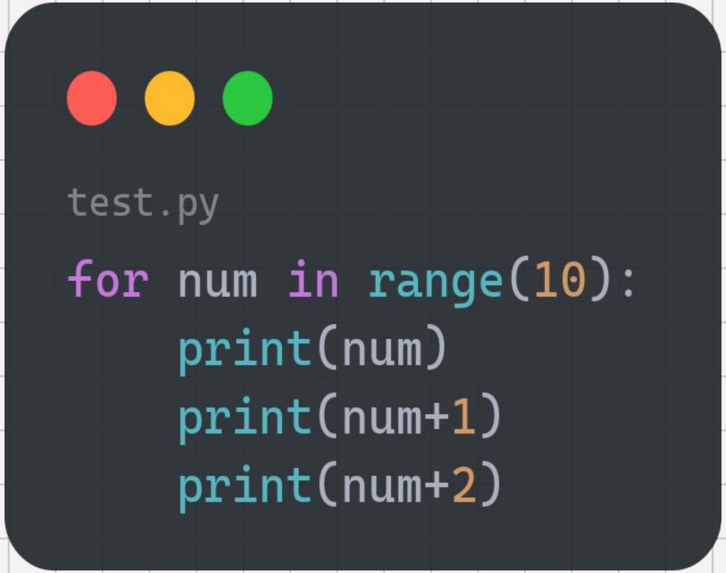


test.py

```
for i in [1,2,3]:
    print(i)
    print(i-1)
    print(i+1)
```

Looping over a range

- We can also loop over a range of values.
- To perform tasks a certain number of times.
- Built-in function range will generate a sequence of integers.
- The sequence starts at 0 and continues to the integer before stop.



test.py

```python
for num in range(10):
    print(num)
    print(num+1)
    print(num+2)
```

The range function

- `range(10)`: start at 0, stop before 10, step size is 1.
- `range(1, 10)`: start at 1, stop before 10, step size is 1.
- `range(1, 10, 2)`: start at 0, stop before 10, step size is 2.
- `range(10, 0, -1)`: start at 10, move towards 0, step size is -1.

While Loops

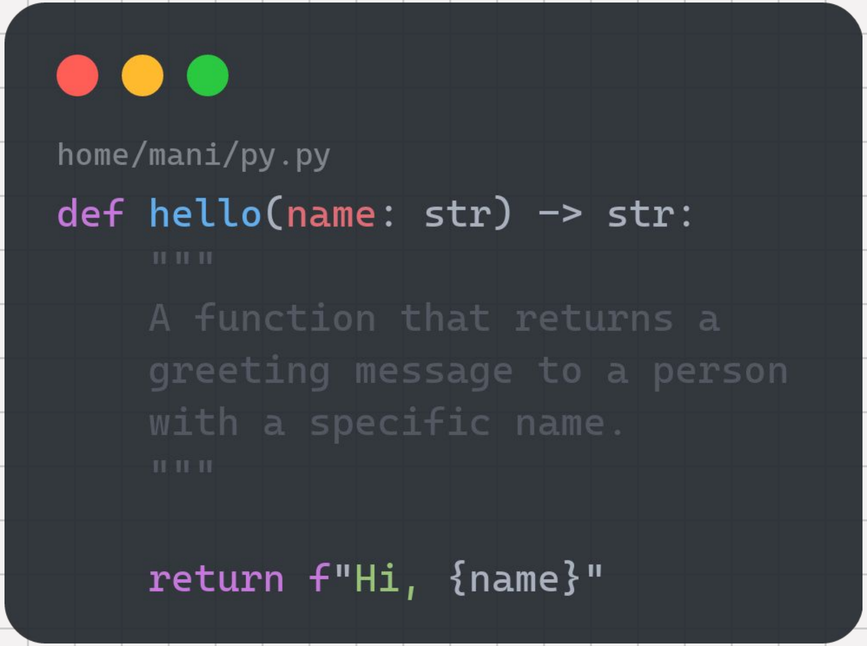
- Works like if, except if body block runs only if the statement is true, but the while body block runs while its statement is true.
- for loops are useful only if you know how many iterations of the loop you need.
- In some situations, it is not known in advance how many loop iterations to execute.
- `while(True)`, `break`, `continue`, etc. work just like the did in C and Java.

Functions!

- Python comes with many built-in functions that perform common operations. Example: `abs(-1)` results in `1`.
- Type convertors (`int()`, `float()`), round (`round()`) and `help()` are functions in Python.
- You can define your own functions with `def [NAME]([PARAMS]):`
- Just like in C, you can have functions that return nothing (`None` return type).
- Here you can pass default args to a function. Example: `def f(a=1):`
- Default parameters must come **after** non-default parameters in the function definition.

Typing and Docstring

- Python uses three double quotes to start and end this documentation; everything in between is meant for humans to read. This notation is called a **docstring**.
- Python's typing in a function header refers to using **type hints** to specify the expected data types of function arguments and the return value.
- If you're not sure what a function does, try calling built-in function `help`, which shows documentation for any function.



home/mani/py.py

```python
def hello(name: str) -> str:
    """
    A function that returns a
    greeting message to a person
    with a specific name.
    """

    return f"Hi, {name}"
```

Modules

- It’s rare for someone to write all of a program alone. It’s much more common, and productive, to make use of the millions of lines of code that other programmers have written before.
- A collection of variables and functions that are grouped together in a single file, is called a **module**. They exist to organize and reuse code efficiently.
- Python comes with a large set of preset modules! You can also install third party modules yourself, just use `pip`!
- You can import other modules in your code! Use `import` and `from … import …`
- If you want to, you may as well write your own modules and contribute to the community or just organize your own projects!

Built-in Modules

- `math`: mathematical operations
- `os`: handles os-level operations
- `logging`: used for better log handling (instead of manual methods, print, …)
- `random`: used for randomized operations (choice, generation, etc.)
- `http`: contains tool-set and utilities for HTTP protocol operations.
- `socket`: used for socket programming (remember AP?)
- `abc`: used to define abstract classes (remember AP?)

Module `__builtins__`

- Python’s built-in functions are actually in a module named `__builtins__`.
- You can see what’s in the module using `help(__builtins__)`.
- If you just want to see what functions and variables are available, you can use `dir` instead.
- Keep in mind, Python loads modules only the first time they’re imported.
- If there is any change to the module, you should either restart the shell or use `imp.reload`.

Running Modules

- Modules can either run **directly** or run when imported by **another python program**.
- Sometimes we want to write code that should only be run when the module is run **directly** and not when the module is imported.
- Python defines a **special string** variable called `__name__` in every module. So you can check if this variable is equal to `__main__`, then execute what you want to execute! (remember main function in C and main method in Java?)

PIP

- Pip stands for "Pip Installs Packages" and is the package manager for Python.
- It is used to install, manage, and uninstall Python packages and libraries from the Python Package Index (PyPI) or other repositories.
- Package Installation: Install any package available on PyPI or compatible repositories. For example: `pip install pandas`
- It’s best to keep your dependencies listed in a `requirements.txt` file, alongside their versions (e.g. `pandas==2.3.0`). Then, you can use:
  `pip install -r requirements.txt`

Sets

- A set is an unordered collection of distinct items. The items aren't stored in any particular order and there are no duplicates.
- A set allows us to store mutable collections of unordered, distinct items.
- Since they are mutable they support methods like `add`, `remove`, and `clear`.
- Just like Java, sets use hashing. Which makes them really fast and convenient!
- You define sets by putting their elements between braces: `a = {1, 2, 3}`
- A set can be created from a list: `a = set([1, 2, 3])`

Set Operations

- `A.add(x)`: adds element `x` to set `A`.
- `A.clear()`: clears all elements from set `A`.
- `A.difference(B)`: returns the difference between sets `A` and `B`.
- `A.intersection(B)`: returns the intersection between `A` and `B`.
- `A.issubset(B)`: returns true if `A` is a subset of `B`. The other way: `A.issuperset(B)`.
- `A.remove(x)`: remove `x` from set `A`.
- `A.union(B)`: returns the union of `A` and `B`.
- `A.symmetric_difference(B)`: returns elements existing in either `A` or `B`, but not both.

Tuples

- Python also has an immutable sequence type called a tuple.
- Tuples are written using parentheses instead of brackets.
- Like strings and lists, they can be subscripted, sliced, and looped over.
- A tuple with one element is not written as (x) but as (x,)
- They can be used to assign multiple values in a single statement, or even perform a variable swap. Example: `s1, s2 = s2, s1`.

Dictionary

- Dictionary is an unordered mutable collection of key/value pairs. (remember maps in Java?)
- They associate a key (like a word) with a value (such as a definition).
- Dictionaries are created by putting key/value pairs inside braces.
- To get, assign, or even update the value associated with a key, we put the key in square brackets, much like indexing into a list.
  Example: `artist[“radiohead”] = [“creep”, “fade out”]`.
- Remember: sets and dictionaries are unordered!
- To remove an entry from a dictionary, use `del d[k]`, where d is the dictionary and k is the key being removed

Class Activity Time!

Which data structures in Python are mutable?

Which data structures in Python are immutable?

Which data structures in Python are ordered?

Which data structures in Python are unordered?

Files

- When you want to write a program that opens and reads a file, that program needs to tell Python where that file is.
- By default, Python assumes that the file you want to read is in the same directory as the program.
- Use `open(f, m)`, with a file path and an opening mode (read `r`, write `w`, appending `a`) to open a file.
- This object that `open` function returns, also keeps track of how much you’ve read and which part of the file you’re about to read next. Then you can use functions like `read`, `readlines`, `write`, `writelines`, etc.

Exceptions

- An exception is an error that occurs during the execution of a program.
- Examples: Division by zero, accessing an undefined variable, invalid file access.
- They interrupt normal program flow.
- They can be handled to avoid crashes.

Common Exceptions

- `ZeroDivisionError`: Raised when dividing by zero.
- `ValueError`: Raised when a function receives an argument of the right type but inappropriate value.
- `FileNotFoundError`: Raised when a file or directory is requested but doesn't exist.
- `KeyError`: Raised when a dictionary key is not found.
- `TypeError`: Raised when an operation is performed on an inappropriate type.

Exception Handling

- You can handle exceptions by using a `try-except` method (remember `try-catch` in Java?)
- Use specific exception types whenever possible and avoid catching generic exceptions unless necessary. (DON'T JUST USE `Exception`!)
- Always include error-handling logic that improves program robustness.
- You can also raise exceptions with `raise` keyword.
  Example: `raise TypeError("What?")`

Classes

- There is another kind of object that is similar to a module: a class.
- A class is how Python represents a type. Class object has the following attributes:
```
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',
'__init__','__init_subclass__', '__le__', '__lt__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__']
```
- Every class in Python, including ones that you define, automatically inherits these attributes from class object. (remember, well… the whole point of Java?)

Those Double Underscores

- Any method (or other name) beginning and ending with two underscores is considered special by Python.
- These methods are typically connected with some other syntax in Python. Use of that syntax will trigger a method call.
- For example, string method `__add__` is called when anything is added to a string (operator overload)

Instance Check

- Function `isinstance` reports whether an object is an instance of a class
- That is, whether an object has a particular type.
- Python has a class called `object`.
- Even classes and functions are instances of `object`. (remember Java?)

Class methods

- To get this to work, we'll define a method called `num_authors` inside Book.
- Book method `num_authors` looks just like a function except that it has a parameter called `self`, which refers to a Book. (remember `this` in Java?)
- We'll write a method that constructs an instance of said object for us. This is a special method and is called `__init__`. (remember constructors?)

home/mani/py.py

```
class Book:
    def __init__(
        self, title: str,
        authors: list[str],
        publisher: str,
        isbn: str,
        price: float
    ) -> None:
        self.title = title
        self.authors = authors[:]
        self.publisher = publisher
        self.ISBN = isbn
        self.price = price

    def num_authors(self) -> int:
        return len(self.authors)
```

Dunder methods

- When we call `print(obj)`, then `obj.__str__()` is called to find out what string to print.
- `__str__` is called when an informal, human-readable version of an object is needed.
- `__repr__` is called when unambiguous, but possibly less readable, output is desired.
- The operator `==` triggers a call on method `__eq__`.

Decorators

- A decorator is a design pattern that allows you to modify or extend the behavior of a function or method without changing its actual code.

- Essentially, a decorator is a function that takes another function as an argument, adds some functionality to it, and then returns the modified version of that function.

- Decorators are commonly used for logging (tracking function calls, arguments, and results), access control (ensuring users are authorized before accessing a function), caching, and other cross-cutting concerns.

home/mani/py.py

```python
def my_decorator(func):
    def wrapper():
        print("before the function is called.")
        func()
        print("after the function is called.")
    return wrapper

@my_decorator
def say_hello():
    print("Hello!")

say_hello()
```