Kotlin for Python Developers Documentation

Release 1.0.0

Paul Everitt <paul.everitt@jetbrains.com>

May 25, 2018

Contents

Kotlin: it's new, it's hot, it's exciting! If you're a Python developer, you might be interested in Kotlin for Android development. If you're a PyCharm user, you might want to write an IDE plugin, using Kotlin.

In both cases, you're looking to to add Kotlin to your programming arsenal. This documentation teaches Kotlin – from the perspective of a Python developer who knows nothing about Kotlin, Java, Gradle, or the JVM.

CHAPTER 1

About This Guide

In no time at all, Kotlin has become an important language to learn, with videos and tutorials and examples and. . . everything you could want.

Unless you're a Python dev. Kotlin is "better Java", but most resources presume you know Java and its ecosystem. It sure would be nice if this shiny new toy was explained in Python terms.

That's what we're doing in this guide. Assuming you know nothing about Java, Gradle, etc., we're going to get you productive in Kotlin and IntelliJ. If you're not a Python developer, you will likely still find this guide useful.

1.1 Why You'll Like This Guide

- Explained in terms of Python
- Presumes no Java knowledge
- Hand-holding and hands-on
- Text, code, and video
- Co-written by Hadi Hariri, a leading spokesperson for Kotlin

1.2 Contributing

We want this guide to be community-oriented and collaborative. Did we do something stupid? File a ticket or even send us a pull request.

CHAPTER 2

Setup

Before jumping into features of the language, let's get ourselves setup. Both Kotlin and Python have sites that let you enter and evaluate code in a browser. For our purposes, let's compare getting a real, local setup for each language.

2.1 The Language

Getting Python and Java installed are similar exercise, but with different obstacles. For Python, while two of the major platforms ship a Python, everybody on earth will beat you over the head if you use "the system Python". For Java, it is not installed by default on most platforms.

Thus, both Python and Java mean an initial, joyful step into the "How do I install it?" thicket.

This is covered in depth elsewhere, so we'll skip it. From a Pythonista's perspective, it's pretty similar. You have to make some choices about which Java distribution, but we have some dirty laundry here as well – 2 vs. 3, python.org vs. platform-y Homebrew etc. vs. Anaconda, PyPy vs. Jython vs. weird experiment of the week.

Call this one a tie.

2.2 The IDE

We're JetBrains: we're going to use the IDE as the entry point for making a project. For both Python and Kotlin, you might make your new project environment from the command line and then open it.

Let's use the IDE make our project, following along with the very-useful Getting Started with IntelliJ IDEA documentation.

2.3 The Project

In PyCharm, you fire it up and have 3 choices: create a new project from scratch, open a directory on disk, or get a clone from a VCS system. In the first, you can choose from some project templates, but you then have the important

part regarding this article:

[screenshot of create new project]

In Python, we're encouraged to put each project in isolation using *virtual environments*. Yet because it is optional, people forget to, and create a world of hurt for themselves. Or they decide to do it, and are confronted with choices and complexity which hurt their brain. PyCharm helps make some of that pain go away:

[screenshot of new interpeter screen]

With two decisions – project type and project interpreter – you now have a new project and environment. PyCharm made some decisions for you, the level of magic is one degree away.

When creating a Kotlin project, you'll confront some technologies that you are unfamiliar with. These are important decisions. Let's walk through them.

Go through step by step: SDK, facets, Gradle, class paths – all from a Python perspective

Virtual environments and *package management* are places where Python is behind. At the same time, the complexity is limited. Java (and for that matter, NodeJS) have stronger stories, but you can't take a step in without feeling pretty dumb about the huge concepts that you don't know.

I wanted to be impartial and call this one against Python. I can't. It's a draw.

2.4 Project Layout

In Python, that's about it. You could, at this point, just make a `.py` file and nobody would complain. Maybe a `.gitignore`.

You could, of course, be a grown-up and make a Python *package*. You'd then be teleported into a world of confusion: `setup.py` vs. `requirements.txt` (vs. `Pipfile` vs. `buildout.cfg`), `MANIFEST.in`, `setup.cfg`, do I put my sources under `project_name` or `src/project_name`. But those are university politics, not state-ordered mandates.

Talk about: src/main/kotlin/someFile.kt and whether that path matters, vs. com.pauleveritt.projectname and how that affects choices.

CHAPTER 3

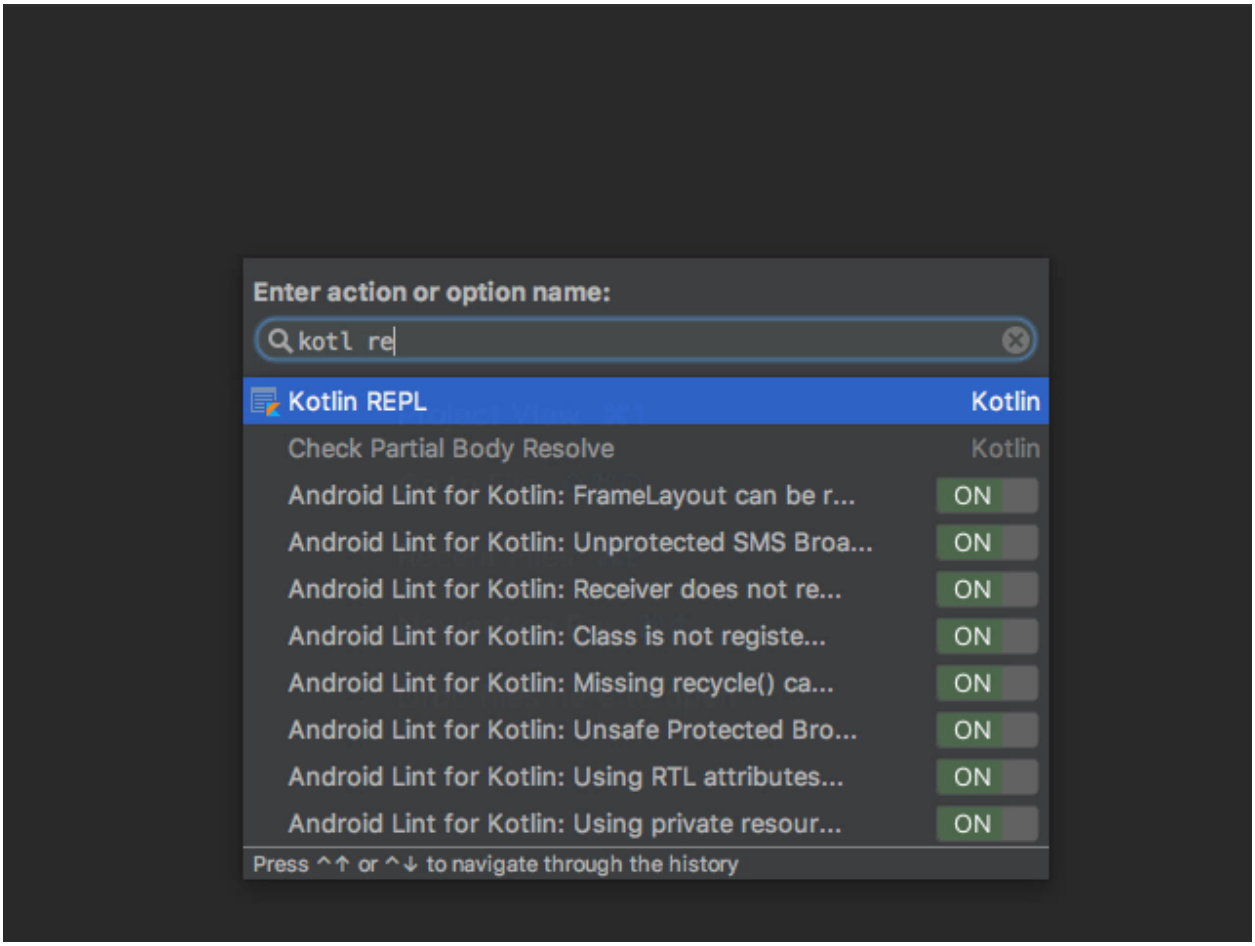
Hello World

Java is installed, our IDE has a project open, we're ready to write some code. In this step we breeze through a light treatment of many Kotlin concepts, all from the Python perspective.

3.1 The REPL

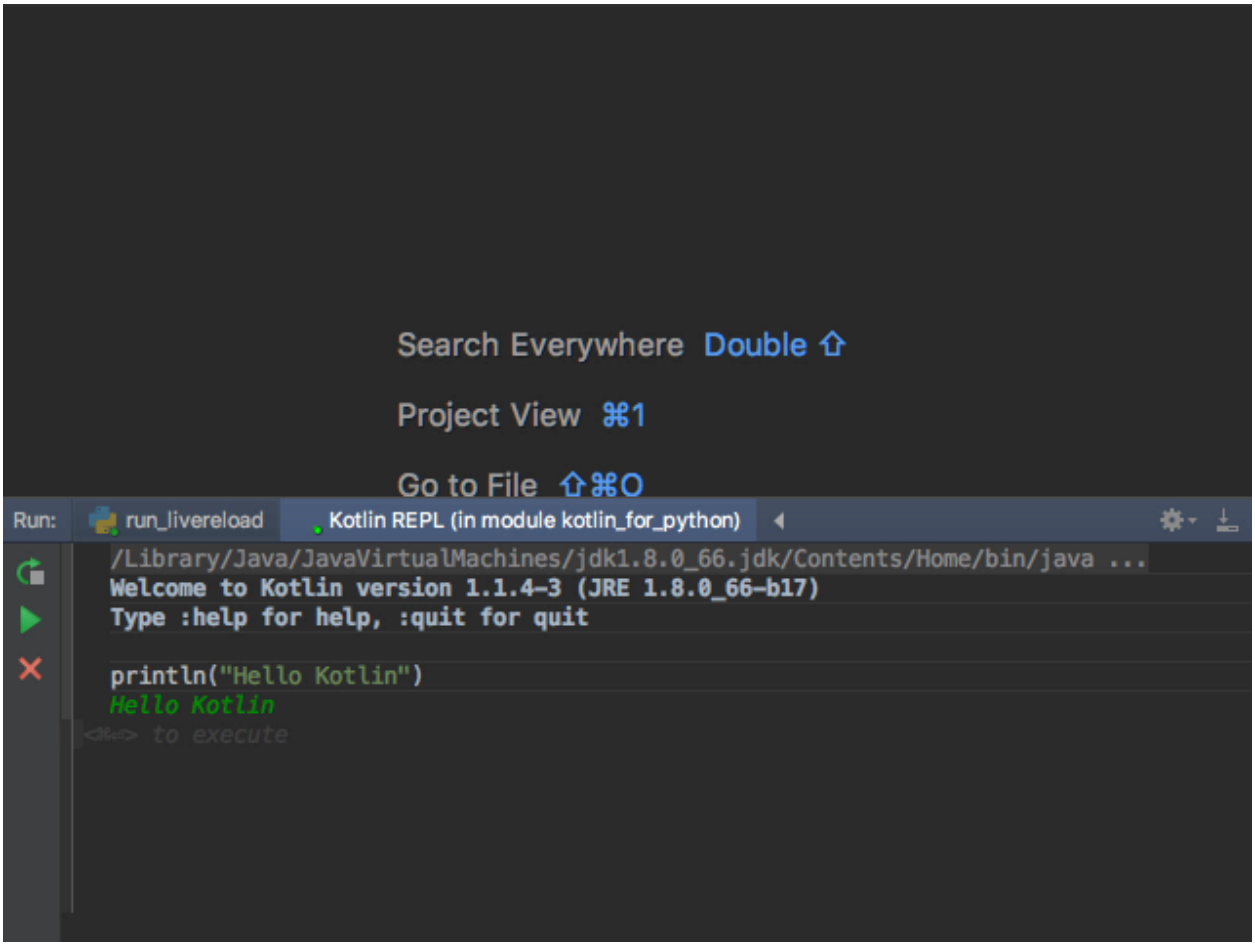
Python has an interactive interpreter, aka REPL, which makes it easy to play around and learn. It's a dynamic language, so this makes sense. As it turns out, Kotlin (in IntelliJ) has a REPL also.

Opening the Kotlin REPL is easy. You can use the `Tools | Kotlin` menu or search for the action:



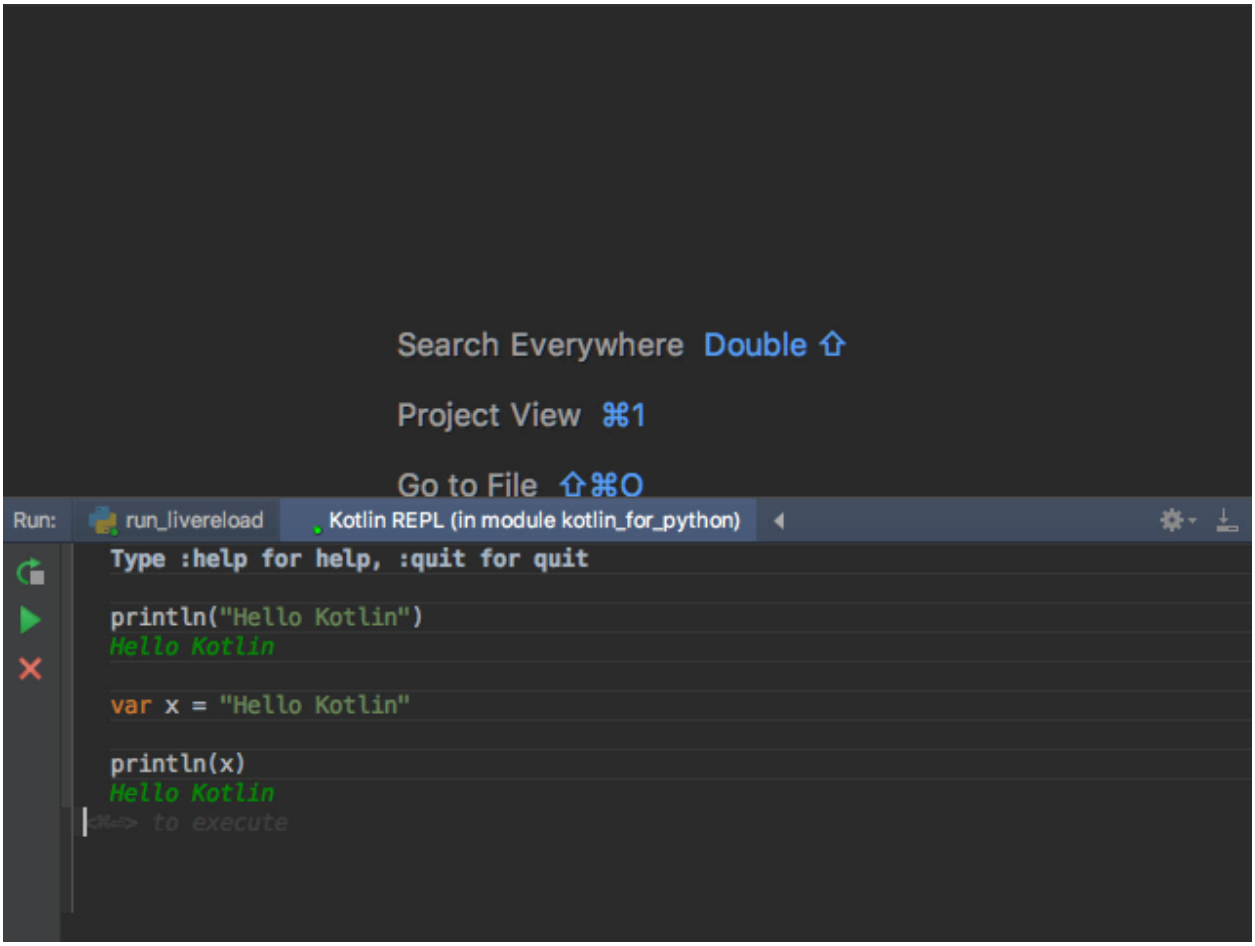
In Python we have the `Python Console` tool window, which opens the Python interpreter in the context of your project. The Kotlin REPL is the same idea.

Let's type in some code:



Here we typed a line of Kotlin code and executed it with `Cmd-Enter` (macOS.) We could have clicked the green play button, which triggers the run action just like `Cmd-Enter`. Kotlin evaluated our line, letting Kotlin/Java do a mountain of machinery behind the scenes.

The REPL can handle multiple lines:



As this is our first foray into Kotlin, let's analyze this small bit of code from the Python perspective:

```
val msg = "Hello Kotlin"
print(msg)
```

- We declare variables with `var` (which allows re-assignment) or `val` (which is like a constant). Python doesn't have `var`.
- Double quotes for strings
- No semicolons!
- A print *function* (like Python 3, but unlike Python 2)

All in all... other than `var`, it's exactly like Python.

Click the red X to close the REPL and let's start writing some Kotlin code.

3.2 First File

In Python, we'd make a `.py` file and start typing in some code. From Python's semantics, there are almost no rules about the file itself – name, location, etc. For example, here is a minimum `hello_world.py`:

```
# Python
print("Hello World")
```

We can start the same in Kotlin. IntelliJ has created a `src` directory for you. Right-click on that and create a file at `src/hello_world/hello_world.kt`:

```
fun main(args: Array<String>) {
    print("Hello World")
}
```

Here's the equivalent Python file to mimic a main function:

```
# Python
def main():
    print("Hello World")


main()
```

Python uses `def` to define a function, Kotlin uses `fun`. We'll talk more about this in *Functions*.

The Kotlin file shows the standard Kotlin "entry point": by convention, the file being executed must have a function named `main` which accepts a single argument, an array of strings. Kotlin itself then calls this main function. This is a *bit* similar to the common (but not required) Python run block. For example, this file in Python might look like this:

```
# Python
def main():
    print("Hello World")


if __name__ == '__main__':
    main()
```

In this Python example, we had to both detect that the module was being run (instead of imported) and then call the appropriate "main" function.

We saw strong typing in the Kotlin function definition. Python of course has typing, but it is at run time and is inferred. (We'll discuss type hinting in the section on *Variables and Types*.)

Time to run this, which really means, compile and execute. If you're familiar with PyCharm run configurations and gutter icons, it's similar. Click the Kotlin symbol in the gutter for line 1 and select `Run`:

[TODO screenshot of running it]

Note: IntelliJ prompted us to `Run 'Hello_worldKT'`. What's `Hello_worldKT`? Answer: To make Java happy, Kotlin generated a Java class behind the scenes.

When you clicked this, there was a lag that you don't get in Python. This the build/compile phases from Java. It's incremental, so it is faster after the first time.

Now that we've run our program, let's breeze through some head-to-head comparisons on a few programming language basics.

3.3 Braces

This is the most obvious point: like most other programming languages, Kotlin delimits blocks with braces. Python uses indentation.

3.4 Quotation Marks

Switching between languages, or even projects, means swinging back and forth between single versus double quotes for strings. For example, TypeScript prefers double quotes, but ReactJS ES6 applications prefer single quotes. And they are both (sort of) JavaScript!

Python's `PEP 8` style guide doesn't prefer one or the other, but most Python projects seem to use single quotes. In fact, Python has triple quoted strings!

```
# Python
hello = 'world'          # best
hello = "world"          # ok
hello = """
   world"""              # triple
```

Java (and Kotlin) use a single quote for a single character value and double quotes for strings. Triple-quotes indicates a raw string:

```
// Kotlin
val c = 'C'              // character
val hello = "hello"      // string
val raw_string = """
    line 1
    line 2
"""
```

3.5 Comments

Kotlin supports the two familiar styles of comments: end-of-line and block comments:

```
val hello = "world"      // Kotlin line comment
/*
    Let's leave out
    these lines
*/
```

IntelliJ (and thus PyCharm) as an IDE makes it easy to comment and uncomment lines and selections with `Cmd-/`.

Python, of course, only uses hash `#` as the comment symbol, with no block comments:

```
#
# Python multiline commments
# have a lot of hashes.

hello = 'world'  # Python comment
```

3.6 Variables

Python doesn't have any special syntax for declaring a variable. You just assign something:

```
# Python
hello = 'world'
```

Kotlin, though, does. In fact Kotlin has two keywords: one to declare a read-only *immutable* value, and one for a *mutable* variable:

```
// Kotlin
val hello = "world"      // Read-only, val == value
var hello = "world"      // Can be re-assigned, var == variable
```

Where's the Java-style type noise? Good news – Kotlin can infer the type. The above is the same as being explicit:

```
// Kotlin
val hello: String = "hello"
```

Also, like Python, you can initialize a variable without assigning it:

```
// Kotlin
var hello
```

Of course with Python 3.6 variable annotations, we can make Python look much more like Kotlin. We cover this in the section on *Variables and Types*.

3.7 String Templates

Python – the "there should be one way to do things" language – actually has *several* ways to do string templates:

```
# Python
msg = 'World'
print('Hello %s' % msg)                  # Original
print('Hello {msg}'.format(msg=msg))     # Python 3 (then 2)
print(f'Hello {msg}')                    # Python 3.6
print(f'Hello {msg.upper()}')            # Expressions
```

Kotlin also has string templates with expressions:

```
// Kotlin
msg = "World"
print("Hello $msg")
print("Hello ${msg.toUpperCase()}")
```

3.8 Functions

Python functions can be very simple:

```
# Python
def four():
    return 2 + 2
```

No curly braces, just indentation. Kotlin's simplest case is pretty close:

```
// Kotlin
fun four(): Int {
    return 2 + 2
}
```

Kotlin adds the curly braces and has to define the return type (which can be omitted if there is no return value.)

But watch this – a function *expression*:

```
// Kotlin
fun four() = 2 + 2
```

Admit it, that's pretty sexy. Function expressions are a big new idea which we'll cover in the section on *Functions*.

Passing in function arguments is straightforward in Python:

```
# Python
def combine(x, y):
    return x + y
```

How does that compare to Kotlin?

```
// Kotlin
fun sum(a: Int, b: Int): Int {
    return a + b
}
```

You have to provide the type information on the function arguments and return value.

3.9 Conditionals

Let's take a quick look at things like conditionals and looping. In Python, an `if/then/else` is straightforward, with use of indentation:

```
# Python
if a > b:
    return a
else:
    return b
```

Kotlin looks quite similar, adding parenthesis (optional in Python) and braces:

```
// Kotlin
if (a > b) {
    return a
} else {
    return b
}
```

We'll cover more on this in *Conditionals*.

3.10 Looping

Let's compare looping over sequences. Simple Python example:

```
# Python
items = ('apple', 'banana', 'kiwi')
for item in items:
    print(item)
```

Here we've created an immutable sequence (a tuple) in `items`. We looped over it in the most basic way possible.

In Kotlin, we have a different construct for making the sequence. Looping is similar, though we use a parentheses after `for`:

```
// Kotlin
val items = listOf("apple", "banana", "kiwi")
for (item in items) {
    println(item)
}
```

In this case we used `println`. In Python, the `print` function always makes a newline unless you ask it not to.

Both Python and Kotlin have rich and interesting control structures, giving both power and terseness. We'll see more in *Sequences and Looping*.

3.11 Classes

Lots to cover later on this, so for now, let's just view the simplest couple of cases. The minimum in Python:

```
# Python
class Message:
    pass
```

In Kotlin:

```
// Kotlin
class Message
```

It's hard to tell which of those have the smaller conceptual density. And who cares – they're both tiny! Let's add a constructor, some variables, and methods. First in Python:

```
# Python
class Message:
    greeting = 'Hello'

    def __init__(self, person):
        self.person = person

    def say_hello(self):
        return f'{self.greeting} {self.person}'
```

This class has a "constructor" with one argument, which is assigned as an instance attribute. The class attribute of `greeting` is used in a method `say_hello` which returns an evaluated f-string.

How about the type hinting flavor for Python 3.5+?

```
# Python
class Message:
    greeting = 'Hello'

    def __init__(self, person: str):
        self.person = person

    def say_hello(self) -> str:
        return f'{self.greeting} {self.person}'
```

Let's see this in Kotlin:

```
// Kotlin
class Message(val person: String) {
    val greeting = "Hello"

    fun sayHello(): String {
        return "$greeting $person"
    }
}
```

That constructor syntax, right in the middle of the class name line, is unusual and cool. It helps to reduce the typing versus Python's constructor. We'll go into more depth on this in the section on *Classes*.

CHAPTER 4

Variables and Types

JavaScript ES6 and TypeScript do (`let` and `const`.)

As we'll see later, Kotlin lets you skip the syntax by inferring type information, but it is still at compile time.

Kotlin will let you know to use a val when you use a var unnecessarily

- Inferred types versus explicit
- The compiler will fail on re-assignment of val
- Scope: top-level versus local

Class variables: properties and fields covered in classes

- Types on variables
- Basic types https://kotlinlang.org/docs/reference/basic-types.html#basic-types

CHAPTER 5

Conditionals

CHAPTER 6

Sequences and Looping

1. 1..5 vs. range(5) (and “a”..”z”)
2. 2 in range(5) (the in operator)
3. if else vs. if then
4. if “expressions” (Python doesn’t have this) var maxValue:Int = if (a > b) a else b but multiple lines (only the last value is used in the block)
5. when (aka switch)...Python doesn’t have this:

```
when (x) {
    !in 1..20 -> println(“x is 1”)
    21,22 -> println(“x is 22 or 23”)
    else -> {
        // Some set of lines in a block
        println(“x is greater than 22”)
    }
}
```

6. etc.

generators, iterators

CHAPTER 7

Functions

Looking at the Kotlin code itself:

- This special `main` function has a contract... it's going to be passed an argument
- We see the first hint of typing. It's mandatory in Kotlin... sort of. In this case we have an array of strings. With Python 3.6 variable annotations for optional type hinting, this would be:

```
import sys
from typing import List


def main(args: List[str]):
    print("Hello World")


if __name__ == '__main__':
    main(sys.argv)
```

- Whereas Python terminates the function line with a colon and uses indentation for the block, Kotlin uses the standard curly braces syntax

Python's weird `if __name__` block is ugly, and reveals a certain something about packaging being added after-the-fact, but shows that Python is ready to just let you do damn fool stupid stuff at module scope. For instance, run your program. Kotlin has a bit of a formal contract to meet when executing an "entry point".

Note: Don't like typing the boilerplate? PyCharm has a Live Template `main` for generating the run block at the bottom. So does Kotlin. We'll show this in the video for this segment.

Kotlin has another syntactic convenience: you aren't required to say that the function returns nothing.

Function expressions

If using Python 3.5+ type hinting, that would be:

```
    def sum(a: int, b:int) -> int:
        return a + b
```

Not too shabby. This will be a recurring point: we'll compare Kotlin not just with "normal" Python, but also against type-hinted-Python.

- Function argument typing and return value typing

CHAPTER 8

Classes

1. Defining a class with a method but no constructor, P has "self"
2. Creating instance of class does NOT require new in either
3. Class variables access in P via self or class name
4. P can assign instance attributes whenever it wants (within __ limitations), change types whenever, no concept of public/private
5. Constructors
6. Binding of constructor arguments to instance attributes (assignment, usage)

    ? - What happens if I don't do var in the constructor? It's unresolved later, but where does it go?

In fact, Kotlin has a rich, multi-layered approach to construction. Our class attribute `greeting` is marked as immutable (and *should* be marked with the optional `private`) as well.

In some ways, Python is clunkier in this example. We have the magic of "dunder" names on important methods, such as the "constructor". The symbol of `self` is sprinkled in to give the instance scope a placeholder. And quite obviously, Kotlin's primary constructor – right after the class name – is terse and doesn't require assigning each value to "self".

Note: Python's `__init__` is called a constructor, but as its name implies, it is actually an initializer. The `__new__` method is the factory.

---

8. Kotlin does some magic behind-the-scenes creation of Java classes named from the file name, because Java needs classes

CHAPTER 9

Packages and Imports

Java Packages, imports, namespaces

Installing Python packages

1. Creating a package, then creating a class in that package
2. Package namespaces
3. Importing from a package/non-package

    ? - What are the magic places that com.hello might find class World? In src, src/main/kotlin ?
4. Installing packages
5. Sharing packages
6. Making “executables”

CHAPTER 10

Indices and tables

- genindex
- modindex
- search