Sekhar, PMP

Installation

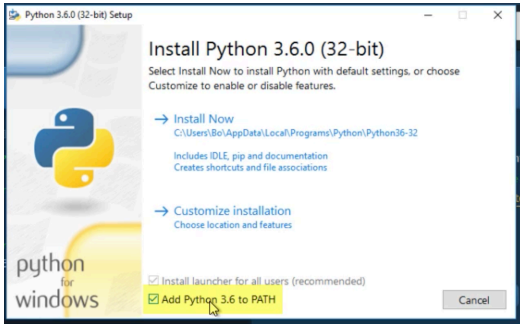
- Download Python 3
- Set the env variables of python for global access
- Download the pycharm community/professional edition
- Install Anaconda
- Set the environmental variables for Anaconda
- Download Java 8 or above
- Set the environmental variables for Java
- Download Scala binaries
- Set the environment variable for Scala
- Download Apache spark

- Installing Python

- www.python.org
- Downloads - select Python 3.7.1 (latest) - The website is smart enough to select the OS
- Once the Download finishes - open the setup file
- Make sure Add Python to path is selected in the installer and select Install Now.
- Go to command prompt and type

python - - version

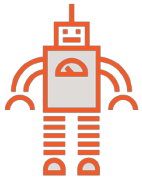
return the python on your machine.



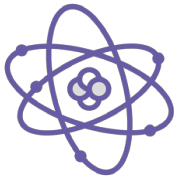
Introduction - What is Python

- Python is a general purpose dynamically compiled langauge. It is compiled on the fly with out needing any special compilation or processing step. Python is compiled and run in all one step
  - Also Java is General Purpose language but it is compiled or converted from code to a language that computers understand in a seperate step.

Using Python



Automation



Science

Desktop

Android

Web

Machine Learning

comprehensions

duck typing

REPL

classes

modules

packaging

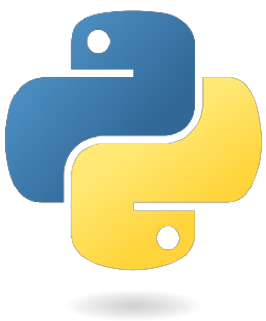
exceptions

resources

standard library

iteration

collections

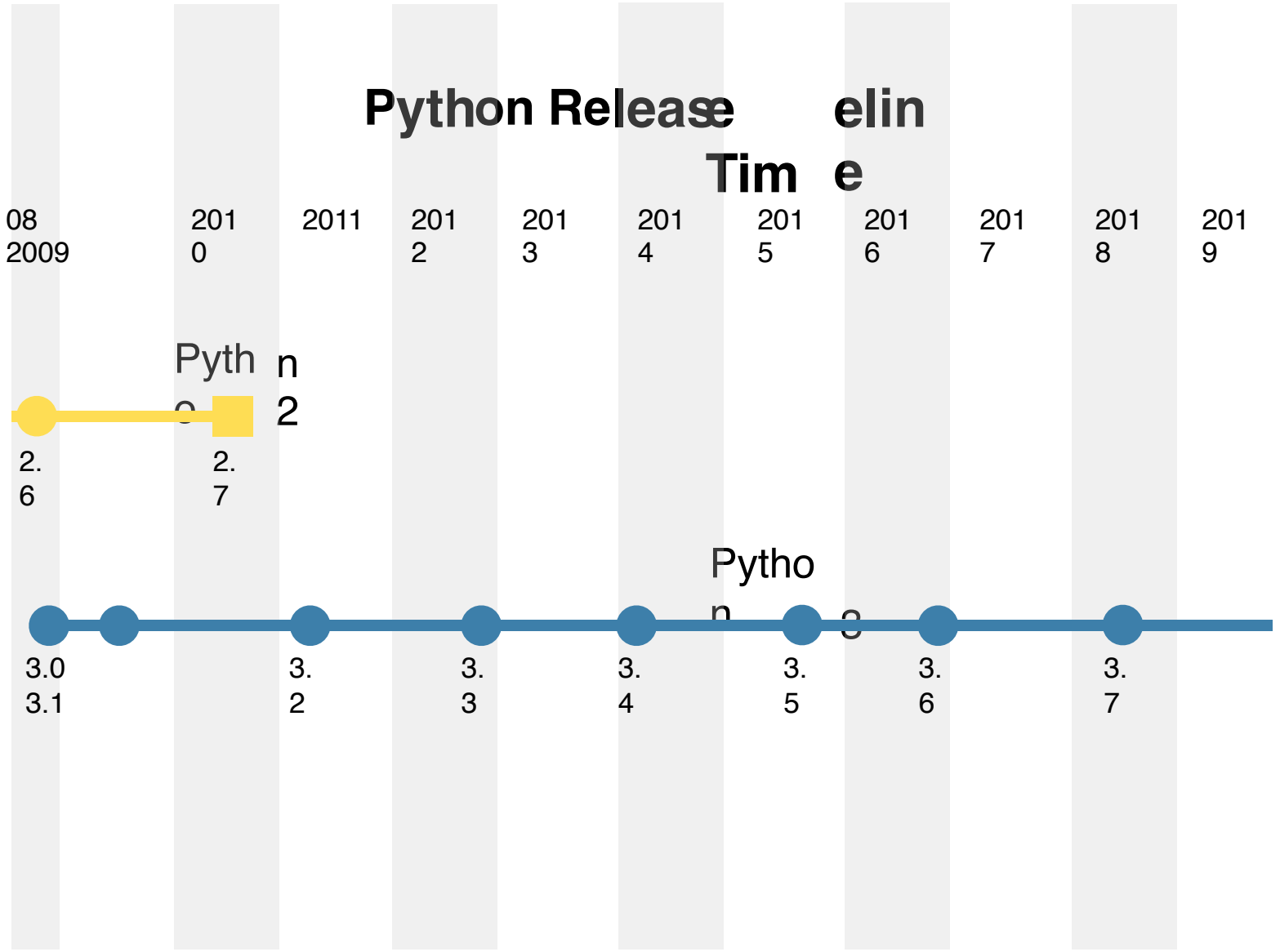


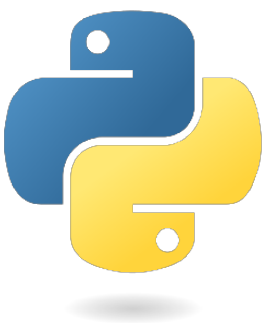
python™

- clear
- readable
- expressive

Python Implementations

Python implementation	written in	runs on
CPython	THE C PROGRAMMING LANGUAGE	native
Jython	Java	JVM
IronPython	C#	Microsoft .NET
pypy	RPython	native & others


Python Release Timeline
08
2009
201
0
2011
201
2
201
3
201
4
201
5
201
6
201
7
201
8
201
9
Python 2
2.
6
2.
7
Python 3
3.0
3.1
3.
2
3.
3
3.
4
3.
5
3.
6
3.
7



python™

Write
Read
Discover
Adventure!

>>> Read

Evaluate

Print

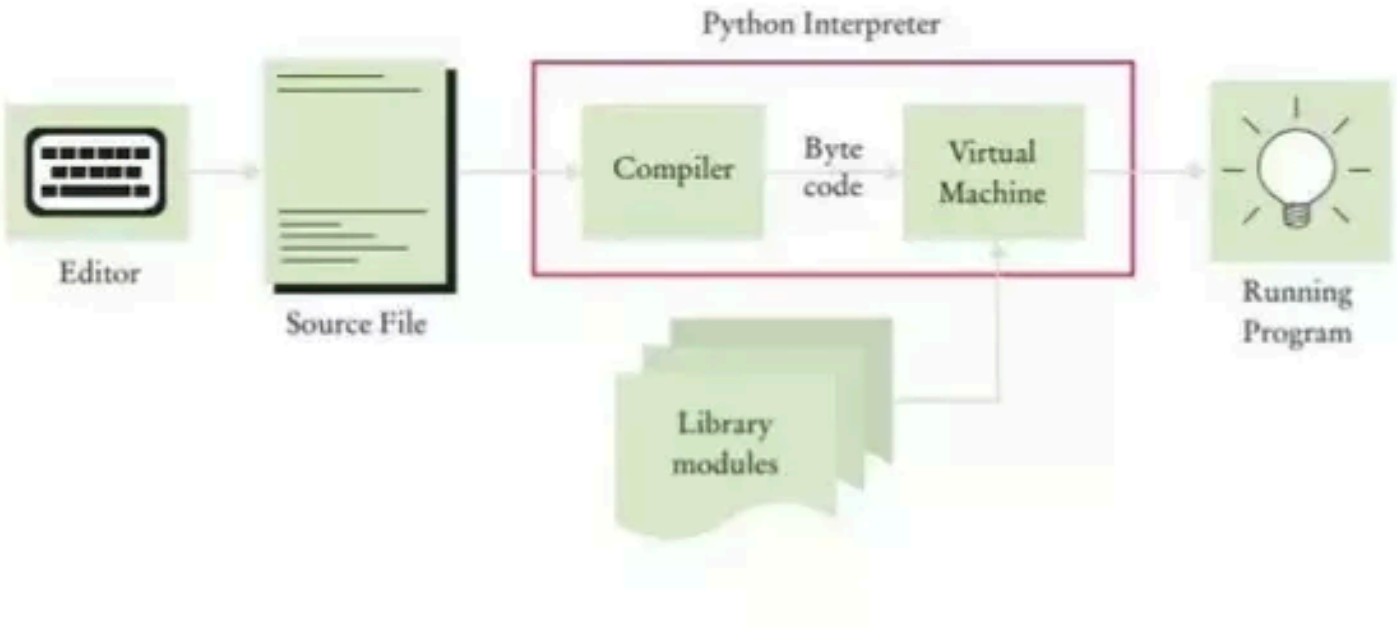
Loop

Python Virtual Machine

- .py vs .pyc

1. .py is the source

How The Python Interpreter Works



Significant Whitespace

1. Requires **readable** code
2. No **clutter**
3. Human and computer can't get **out of sync**
4. Prefer **four spaces**
5. **Never** mix spaces and tabs
6. **Be consistent on consecutive lines**
7. **Only deviate to improve readability**

Syntax and Style

Every programming language has specific words and rules for using them which we call syntax. Some languages also have rules about how the code should look. Python has both of these!

```
def answer_the_phone():
____greeting = "Hi, this is Kenneth. Who is this?"
    print(greeting)
```

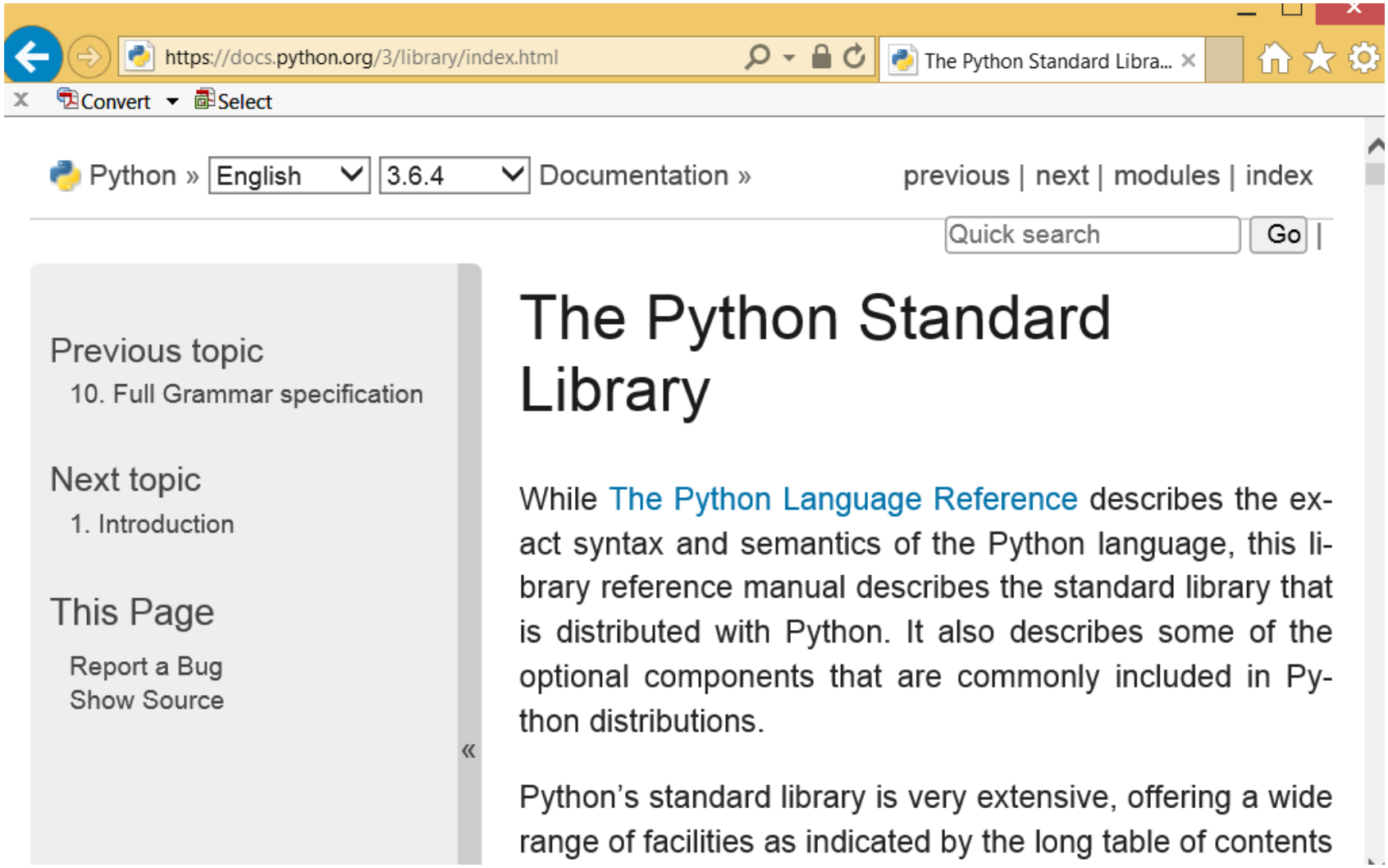
- http://legacy.python.org/dev/peps/pep-0008/

Blocks start with a colon.

Variable and function names should all be lowercase and use underscores instead of spaces.

you can't use spaces in variable or function names.

You also can't use hyphens, or Python will try to subtract things


https://docs.python.org/3/library/index.html
The Python Standard Libra...
Convert
Select
Python »
English
3.6.4
Documentation »
previous | next | modules | index
Quick search
Go
Previous topic
10. Full Grammar specification
Next topic
1. Introduction
This Page
Report a Bug
Show Source
The Python Standard Library
While The Python Language Reference describes the exact syntax and semantics of the Python language, this library reference manual describes the standard library that is distributed with Python. It also describes some of the optional components that are commonly included in Python distributions.
Python's standard library is very extensive, offering a wide range of facilities as indicated by the long table of contents

Python Standard Library

string	numbers	pickle
difflib	fraction	marshal
textwrap	random	configparser
struct	itertools	hashlib
codecs	operator	logging
datetime	functools	time
heapq	filecmp	threading
weakref	tempfile	concurrent
copy	glob	email
pprint	linecache	asyncio

etc., etc., etc!

Variables

- Variables always point to data, and programming is mostly about working with that data.
- The data is of different types. – Similar to species in animals and plants.
- The type of the value doesn’t affect the variable at all. Not all values are variables.
- Unlike other languages You don’t have to declare Python Variable before you use them, they’re on Demand. Python has no restrictions on the variables. Python just cleans up any unused names or values when you delete things

what is a Variable

Variables are names that point to particular pieces of data. In Python, we create a variable like this:

variable_name = "variable value"

Not all values will need quote marks around them, but the format is always the same. The variable name on the lefthand side of the single equals sign and the value is on the right.

Variables - Data Types - Naming

Creating a variable is always done the same way:

```
variable_name = "variable value"
```

And you can always delete a variable using:

```
del variable_name
```

- Luckily, Python doesn't mind if we use long variable names.Name things so that you'll remember what they are.The more obvious your variable and function names are, the more quickly you'll be able to code and make cool things.
- Python also, of course, lets us delete variables. Most of the time pythons built in garbage collection, which cleans up behind you, will do a good enough job to keep everything moving smoothly.

Scalar types and values

int 42	arbitrary precision integer
float 4.2	64-bit floating point numbers
NoneType None	the null object
bool True, bool False	boolean logical values

Relational Operators

==	value equality / equivalence
!=	value inequality/ inequivalence
<	less-than
>	greater-than
<=	less-thanor equal to
>=	greater-than or equal to

Comparison Operators

Python has seven comparators, or comparison operators. Each of them returns `True` or `False`. Here they are:

Operator	Comparison
`==`	A is equal to B
`!=`	A is not equal to B
`<`	A is less than B
`>`	A is greater than B
`<=`	A is less than, or equal to, B
`>=`	A is greater than, or equal to, B
`is`	A has the same memory address as B

Numbers

The two built-in types of numbers in Python are `int` and `float`. Ints are whole numbers and floats are decimal numbers.

Both number types support addition (`+`), subtraction (`-`), multiplication (`*`), division (`/`) and more.

Division *always* results in a float, even if the result would be a whole number (e.g. `10 / 2` will give you `5.0`).

`round()` is a built-in function that will round a float to the nearest whole number.

Dividing by zero will result in a `ZeroDivisionError` exception.

You can create integers using the `int()` function. You can create floats using the `float()` function. Using `int()` on a `float` is **not** the same as using `round()` on one.

String

- Str - immutable sequence of Unicode code points. immutable meaning that once you've constructed a string you can't modify its contents.

- You can use single or double quotes but be consistent

- multiline strings are delimited by three quote characters rather than one.

Strings

Strings have to start and stop with the same quote symbol, either single (`'`) or double (`"`) or triple (`'''` or `"""`) quotes. You also don't want to use that same quote symbol *inside* the string unless you escape with with backslash (`\`).

Strings can be combined with the plus sign (`+`) which is known as **concatenation**.

Strings can also be combined using string formatting which uses two braces (`{}`) as placeholders and the `.format()` method. You provide a bit of data to `.format()` for every placeholder in your string. For example:

"My {} is {}".format("name", "Sekhar")

This string has two placeholders so we have to give two new things to `.format()`.

You can also multiply strings by an integer. This will result in the string's content being repeated as many times as the value of the integer. `"hi" * 3` would create `"hihihi"`.

And, finally, you can create a string from another value by using the `str()` function.

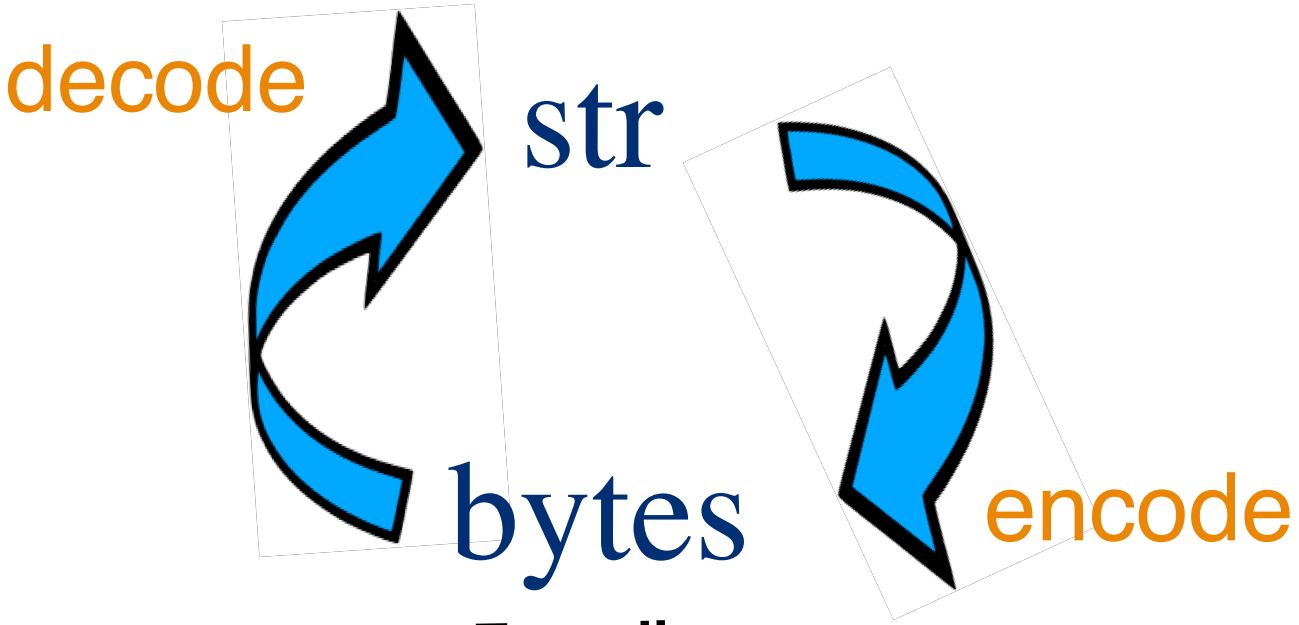
Boolean

The `bool()` function will tell you whether something (a variable, a comparison (next video!), or anything else) evaluates to true or false. You won't find this used a lot in every day Python programming but it's super-handy for testing your assumptions when you're still learning.

Escape Sequences

Sequence	Meaning
`\newline`	Backslash and newline ignored
`\\`	Backslash (\)
`\'`	Single quote (')
`\a`	ASCII Bell (BEL)
`\b`	ASCII Backspace (BS)
`\f`	ASCII Formfeed (FF)
`\n`	ASCII Linefeed (LF)
`\r`	ASCII Carriage Return (CR)
`\t`	ASCII Horizontal Tab (TAB)
`\v`	ASCII Vertical Tab (VT)
`\ooo`	Character with octal value ooo
`\xhh`	Character with hex value hh
Only recognized in string literals	
`\N{name}`	Character named name in the Unicode database
`\uxxxx`	Character with 16-bit hex value xxxx
`\Uxxxxxxxx`	Character with 32-bit hex value xxxxxxxx

Converting Strings and Between Bytes



Encodings

Conditional Statements

```
if expr:
    print("expr is True")
else:
    print("expr is False")
```

else

And, finally, we have the `else:` block for both loops. If the loop ends naturally, meaning it doesn't have a `break` triggered and no exceptions happen, the loop's `else` block will happen, if one exists. The `else` block is entirely optional, just like with `if`.

```
for num in [2, 4, 6]:
    print(num*10)
else:
    print("That's all of the numbers, multiplied by 10.")
```

This loop can't throw an exception and doesn't have a `break` in it, so the `else` will always happen.

Condition - comparison

Python `if` blocks always start the same way:

if a < b:

    print("A is tiny!")

It's always the keyword `if` followed by the condition. In this case, the condition is `a < b` but your conditions will be different. Then comes a colon (`:`), a return, and the body of the `if` is indented.

If you want something to happen when the condition is `False`, you'd put that into an `else:`block, which is just the word `else` followed by a colon.

If you need another possible condition, use the `elif` block, like so:

elif a == b:

    print("A and B are equivalent!")

The `elif` block is built exactly like the `if` block but with `elif` instead of `if`. All of your `elif`blocks must come before any `else` blocks.

Lists

You can create a list with the `list()` function or using two square brackets (`[]`).

You can use the plus sign to add items into a list so long you add two lists together (e.g. `[1, 2, 3] + [4, 5]` would create `[1, 2, 3, 4, 5]`. `[1, 2, 3] + 4` would create a `TypeError`). You can multiply a list by an integer and get back the content of the list as many times as the value of the integer (e.g. `[1, 2] * 2` would produce `[1, 2, 1, 2]`).

You can use the `.append()` method to add **a single item** to the end of a list.

You can use the `.extend()` method to add **every item** from one list to another list.

You can use the `.remove()`method to remove a particular value from a list.

Split

When we want to break a string into a list based on something that repeats in the string, we use the `.split()` method. By default, `.split()` splits the string on whitespace (i.e. spaces, tabs, newlines, etc). If you want to split based on something else, give that something else to the `.split()` method.

"Hello there".split() produces ["Hello", "there"].

"name;age;address".split(";") produces ["name", "age", "address"].

Join

The `str` type's `.join()` method lets us combine all of the items in a list into a string with a particular string between each pair of items.

my_favorite_colors = ["green", "black", "silver"]

my_string = "My favorite colors are "

my_string + ", ".join(my_favorite_colors)

The above would produce "My favorite colors are green, black, silver".

You **cannot** join things that aren't strings. Doing `", ".join(5, 10, 15)` will give you an exception.

Growing Lists

- Methods used

`.append(<value>)` - Add a new value onto the end of a list.

`.extend(<iterable>)` - Make a list longer by adding on the members of another iterable.

`.insert(<index>, <value>)` - Add a value to a list at a particular index.

You can, of course, also add lists together with the `+` operator. To do this in place, you'd use the increment operator `+=`.

Index

Indexes count **up** from 0 when you start on the lefthand side. In the string `"hello"`, the `"h"` is at index 0, the `"e"` is at 1, the first `"l"` is at 2, and so on.

Indexes count **down** from -1 if you start on the righthand side. Using the string `"hello"`, we could get the `"o"` at the index -1, the second `"l"` at -2, etc.

Both strings and lists have a `.index()` method that will give you the index for a particular value. Using our old friend `"hello"`, if we wanted to find the index for the `"e"`, we could do `"hello".index("e")`.

`.index()` only gives the index for *the first occurrence* of whatever you searched for.

Deletion

`del my_list[0]` and `my_list.remove("a")` do two very different things so be careful when you use each one. `del` deletes an item at a particular index. `.remove()` deletes the first instance of the value you provide it.

keyword - in (Containment)

`in` returns whether or not a value is inside of a container. We can use this to see if, for example, a smaller string is in a bigger string or if a certain item is in a list.

Examples

"java" in "javascript" would give back True

`5 in [3, 4, 1]` would give back `False`

while loops

```
while expr:
    print("loop   while expr is True")
```

expr is converted to bool as if by

the bool() constructor

Loops - For

Python has two loop types, `for` and `while`.

`for` loops run a certain number of times and then end themselves. `while` loops, on the other hand, run until their condition, like an `if` has, turns `False`. If it helps, you can think of `while`loops as repetitious `if`s.

For loop

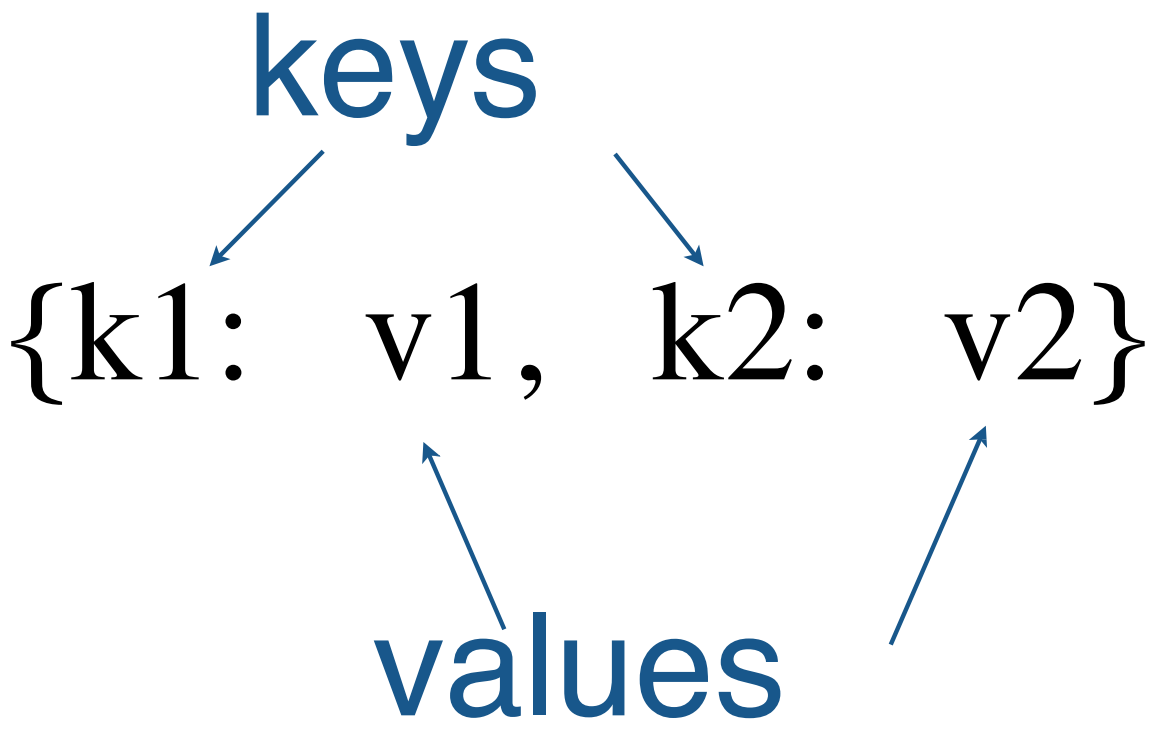
```
numbers = [1, 2, 3, 4]

doubles = []

for number in numbers:

    doubles.append(number*2)
```

This loop will run four times because there are four items in the `numbers` list. On each iteration of the loop, we're calling the value at the current index `number`.

For-Loop Syntax

```
for item in iterable:
    ...body...
```

Dict Literals



Dictionaries

rDictionaries, or dicts, are usually constructed with the `{}` literals. You provide a key, a colon, and then a value.

course = {"title": "Python Collections”}

You can access the keys by using `[]` and the key name.

course[“title"]

"Python Collections”

Dictionaries Management

- .update() - Pass in a dictionary of keys and values to create or update in a single step.

You can override a single key by assigning a new value to it. And you can delete a key by using `del` and the key's name.

- try

`.pop(<key>)` - like lists, dicts have `.pop()`. It'll return the key's value to you and then delete the key.

`.popitem()` - similar to `.pop()` but instead of returning just the value, returns you a tuple (more in the next stage!) with the key and the value. Also, this doesn't take any arguments, you get a random key/value pair!

`.clear()` - need to quickly empty out a dictionary? This method is your tool of choice, then.

Functions

- Function – lines of code that you can trigger at any point. Functions are another type of value to

```
def my_function_name():
    print("this is the function body”)
```

Notice that the function's first line ends with a **colon** and the **function body**, the line or lines that belong to the function is **indented**

Functions are another type of value and we can use anywhere we would use another variable

Objects

An object is another value type. All of our values are actually objects.

Objects represent something with concrete attributes.

Objects also have functions that belong to them. We call these functions, "methods", and they represent actions that the object can do.

Errors

None of us are perfect which means we'll all eventually encounter bugs and errors in our code. Since it's going to happen, we should learn how to prevent and recover from them.

`NameError` - You tried to use a name (a variable, function, or other object) that doesn't exist.

`TypeError` - You tried to use a piece of data that's not the expected/right type *or* you tried to do something that a type of data can't do.

`SyntaxError` - You tried to use some invalid Python.

While

While loop

```
start = 99
while start:
    print("{} bottles of milk on the wall, {} bottles of milk.".format(start, start))
    print("Take one down and pour it on some cereal.")
    start -= 1
    print("{} bottles of milk on the wall.".format(start))
```

This `while` loop will print out something like the traditional "99 bottles" song many of us used to annoy our parents on road trips. The loop stops when `start` becomes something `False`, which will happen when it's reduced to 0.