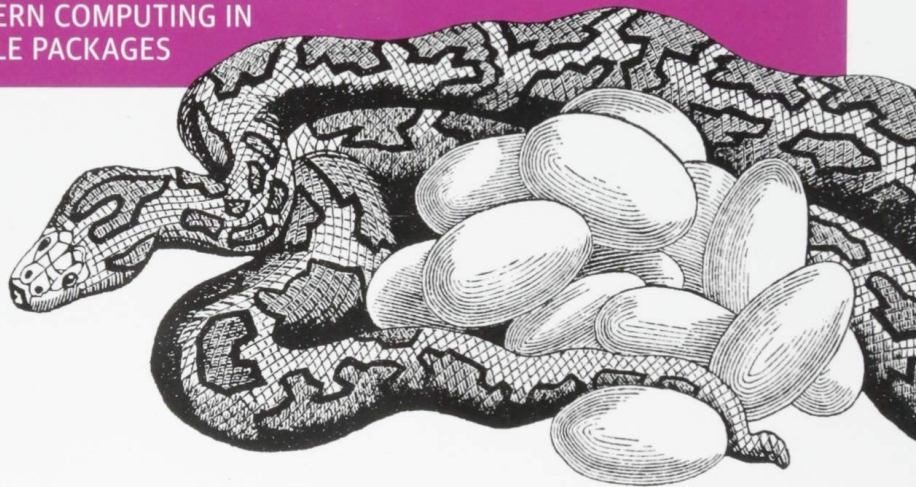


O'REILLY®

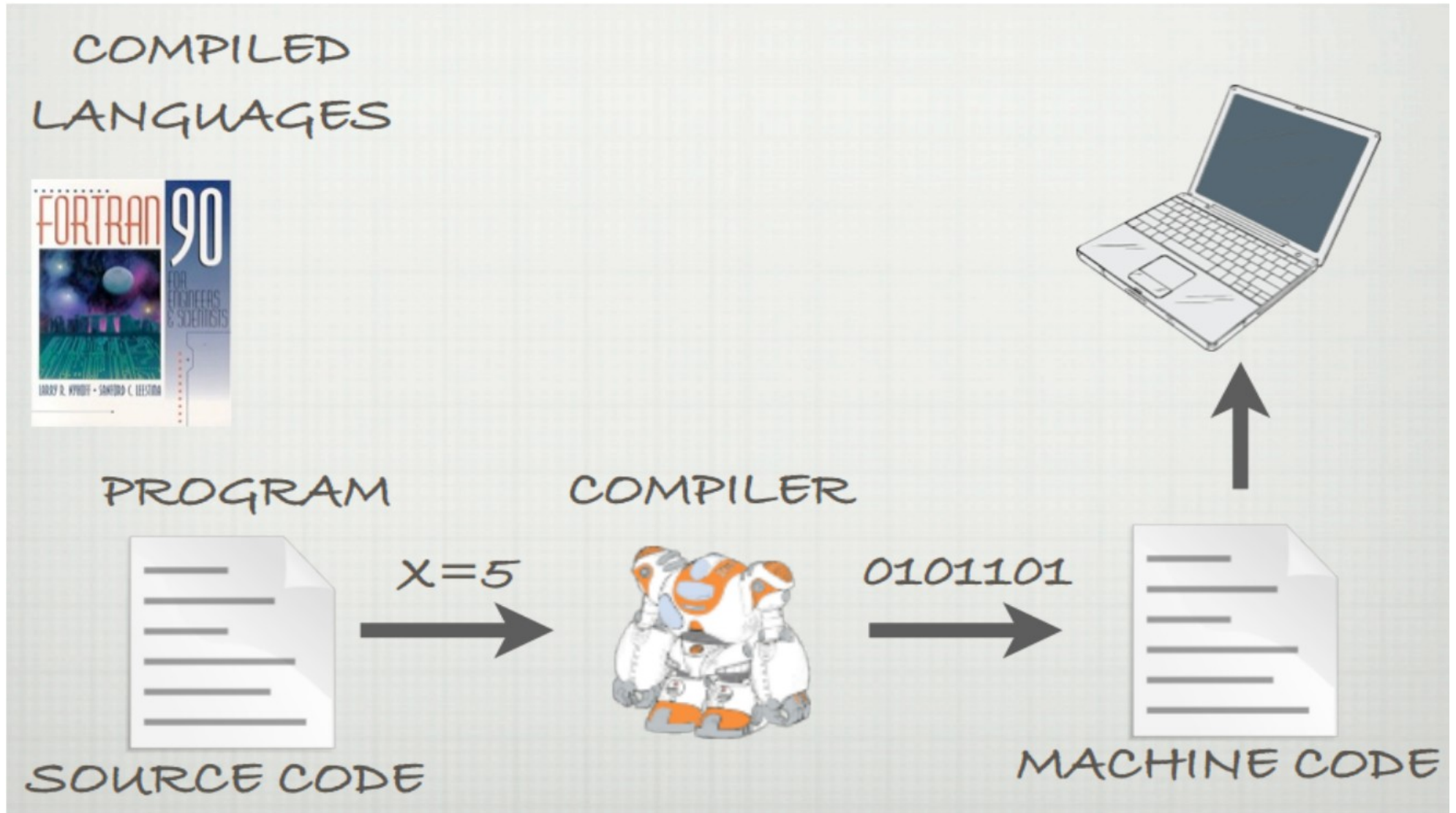
Introducing Python

MODERN COMPUTING IN
SIMPLE PACKAGES



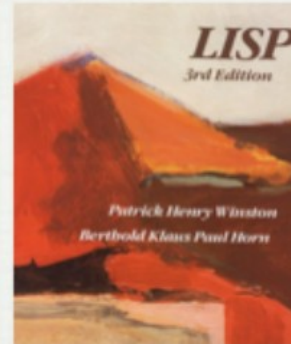
Bill Lubanovic

Interpreters and compilers



Interpreters and compilers

INTERPRETED
LANGUAGES

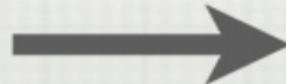


PROGRAM

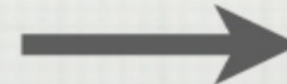
INTERPRETER



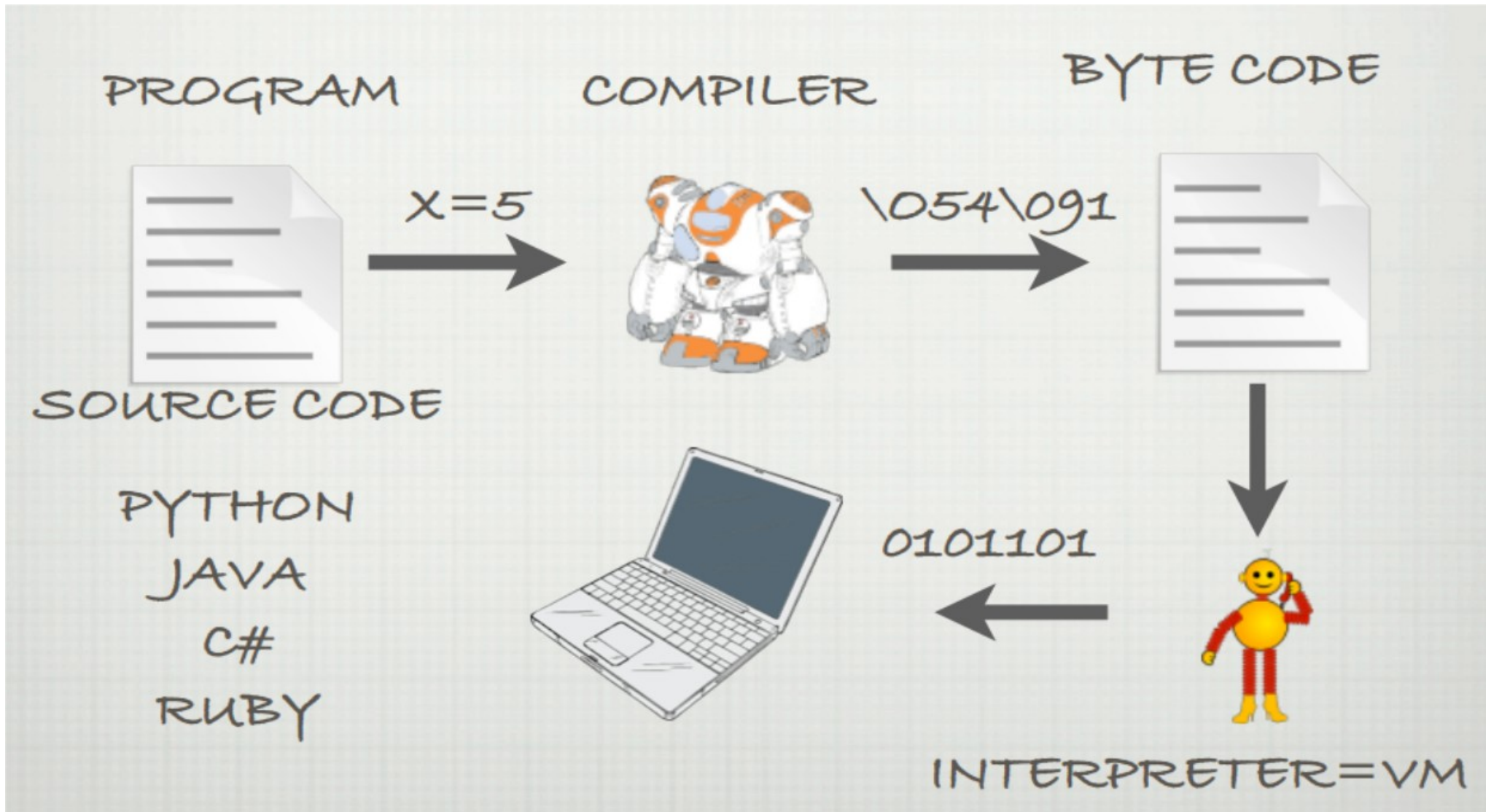
$x=5$



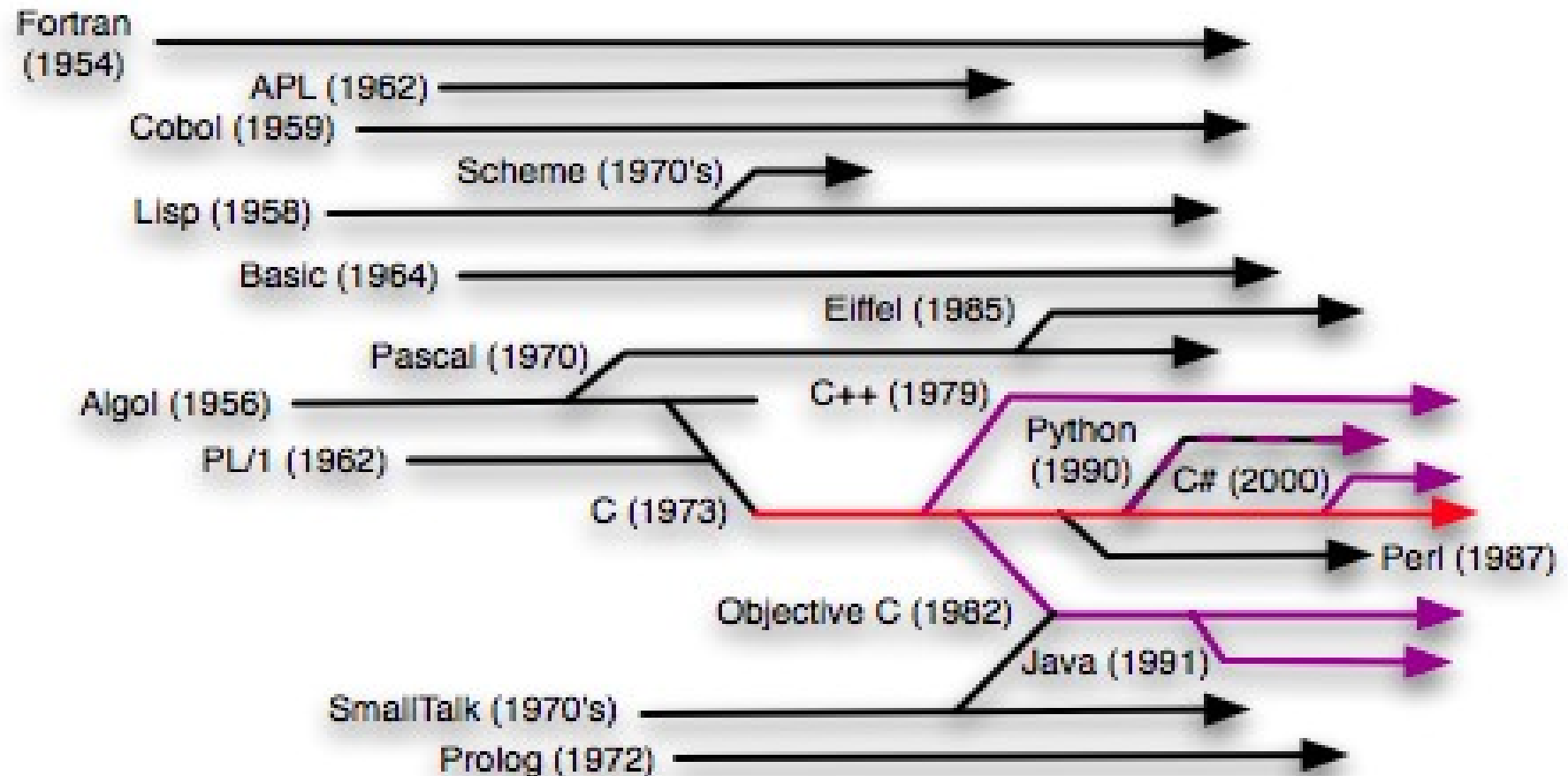
0101101



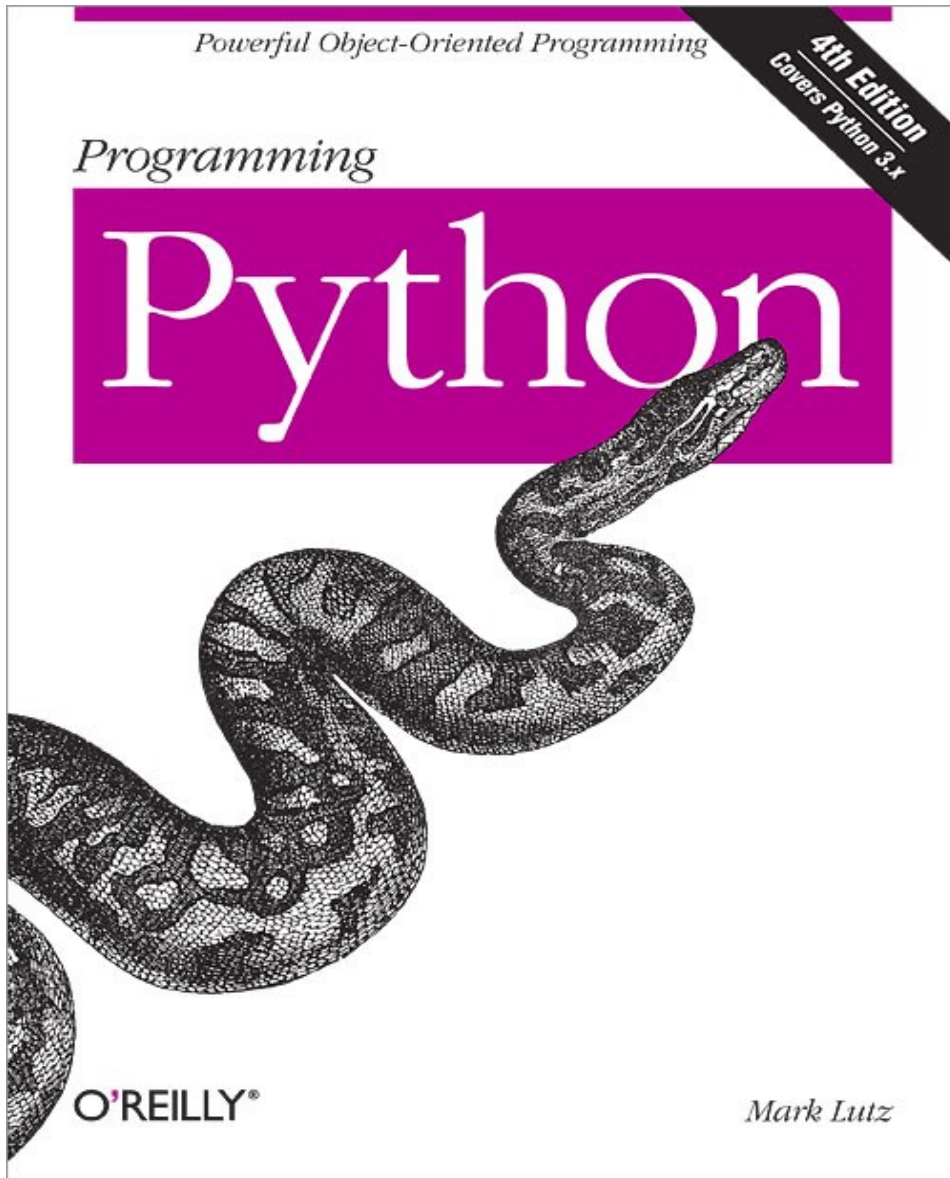
Interpreters and compilers



Programming languages



Python



<https://www.python.org/>

<https://docs.python.org/3/>

<https://docs.python.org/3/tutorial/>

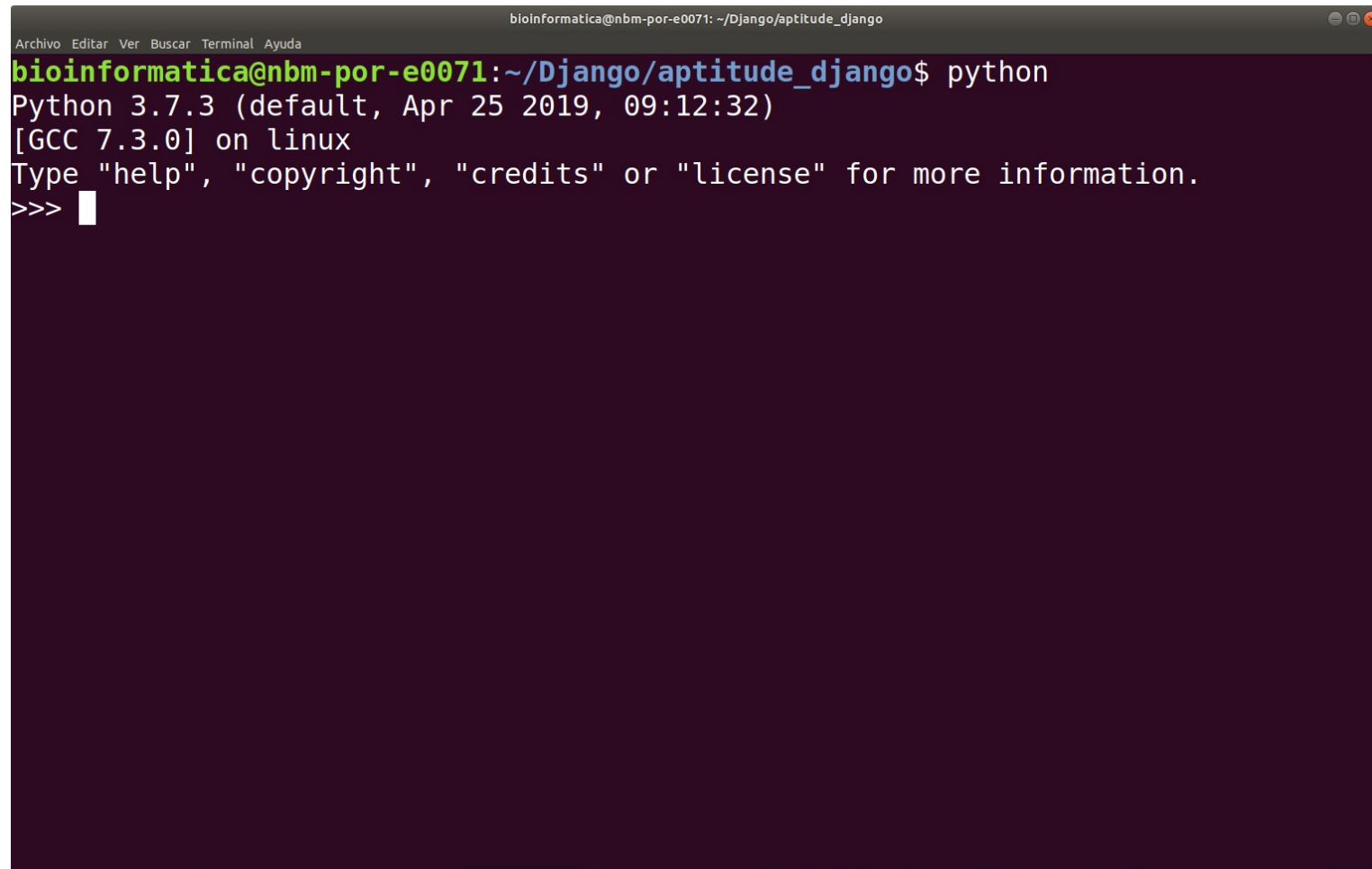
<https://www.python.org/dev/peps/pep-0008/>

<http://www.diveintopython3.net/>

<http://www.learnpython.org/>

<https://realpython.com/>

Python

A terminal window with a dark background and light text. The title bar at the top reads 'bioinformatica@nbm-por-e0071: ~/Django/aptitude_django'. The menu bar below it contains 'Archivo', 'Editar', 'Ver', 'Buscar', 'Terminal', and 'Ayuda'. The main content area shows the command 'python' being executed, followed by the output: 'Python 3.7.3 (default, Apr 25 2019, 09:12:32)', '[GCC 7.3.0] on linux', and a prompt to type 'help', 'copyright', 'credits', or 'license' for more information. The prompt '>>>' is followed by a white cursor block.

```
bioinformatica@nbm-por-e0071: ~/Django/aptitude_django$ python
Python 3.7.3 (default, Apr 25 2019, 09:12:32)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```



```
1 # file.py
```

```
2 print("Hello World")
```


Python

- **Everything in Python is an object, and almost everything has attributes and methods.**
 - **Object:** A single software unit that combines attributes and methods
 - **Attribute:** A "characteristic" of an object; like a variable associated with a kind of object
 - **Method:** A "behavior" of an object; like a function associated with a kind of object
 - **Class:** *Code that defines the attributes and methods of a kind of object (A class is a collection of variables and functions working with these variables)*

Python

- Python scripts have extension .py
- You can run them from the command line invoking the python interpreter

```
python my_script.py
```

Python

```
#!/usr/bin/python
```

```
#
```

```
# Let's calculate diameter, circumference and area for a circle of given radius
```

```
#
```

```
import math
```

```
r = input("What is the radius of your circle (in cm)? ")
```

```
print("The diameter of your circle is {} cm".format(2*r))
```

```
print("The circumference of your circle is {} cm".format(2*math.pi*r))
```

```
print("The area of your circle is {} cm2".format(math.pi*pow(r,2)))
```

Control statements

- Make choices based on conditions – to selectively execute certain portions of the code
 - Use `if` to execute code based on a condition
 - Use `if-else` to make a choice based on a condition
 - Use `if-elif-else` structures to make a choice based on a series of conditions

Python

- Python has a way to put definitions in a file. Such a file is called a module
- Definitions from a module can be imported into our code by using the import statement

```
>>> import math
```

- The built-in function `dir()` is used to find out which names a module defines. It returns a sorted list of strings:

```
>>> dir(math)
```

- You can get `help()` on any function name to see how to use it.

```
>>> help(math.cos)
```


Control statements

Conditional Loops

while *logical_expression*:
statements...

```
>>> i = 0
>>> while i < 5:
...     print(i)
...     i += 1
...
0
1
2
3
4
```

Iterative Loops

for *individual_item* in *iterator*:
statements...

```
>>> names = ["chris", "iftach", "jay"]
>>> for name in names:
...     print(name)
...
chris
iftach
Jay
```

Generating Random Numbers

- Great for simulations and games
- Random numbers generated by computer are not truly random but pseudorandom, generated by formula; complex but predictable pattern
- Need to use a module (random)

```
import random
```

```
random.randrange(n) # generates random number  
# from 0 to n - 1
```

Guess My Number Game

```
from random import randrange
print("Welcome to 'Guess My Number'!")

print("I'm thinking of a number between and 100")
print("Try to guess it in as few attempts as possible")

c=0
n=randrange(1,101)
while True:
    a = input("Take a guess: ")
    c+=1
    if (a>n):
        print("Lower...")
    elif(a<n):
        print("Higher...")
    else:
        print("You guessed it!")
        print("and it only took you {} attempt(s)".format(c))
        break
```

String functions

```
>>> len("GATTACA")  
7
```

← Length

```
>>> print "GAT" + "TACA"  
GATTACA
```

← Concatenation

```
>>> print "A" * 10  
AAAAAAAAAAAA
```

← Repeat

```
>>> "GAT" in "GATTACA"  
True
```

← Substring tests

```
>>> word = "GATTACA"  
>>> word[1:4]  
ATT
```

← Assign a string slice to a variable name

String Methods

```
quote = "I think there is a world market for maybe five computers."
```

```
>>>quote.upper()
```

```
I THINK THERE IS A WORLD MARKET FOR MAYBE FIVE COMPUTERS
```

```
>>>quote.lower()
```

```
i think there is a world market for maybe five computers.
```

```
>>quote.title()
```

```
I Think There Is A World Market For Maybe Five Computers.
```

```
>>quote.replace("five", "millions of")
```

```
I think there is a world market for millions of computers.
```


String Methods

```
word = "engineering"  
>>> word.startswith("e")  
True  
>>> word.endswith("e")  
False  
>>> word.count('e')  
3  
>>> word.find("ne")  
4  
>>> "ne" in word  
True
```

Indexing and slicing

0	1	2	3	4
i	n	d	e	x
-5	-4	-3	-2	-1

- Use brackets and position number to index `index[3]`
- Positive position numbers: starts at 0; ends with the length of a sequence - 1
- Negative position numbers: starts at the end of sequence with position number: -1
- Attempt to access beyond last position results in error

Indexing

```
>>> word = "engineering"
```

```
>>> len(word)
```

```
11
```

```
>>> word[0]
```

```
'e'
```

```
>>> word[1:5]
```

```
'ngin'
```

```
>>> word[5:]
```

```
'eering'
```

```
>>> word[-3:]
```

```
'ing'
```

```
>>> word[:3]
```

```
'eng'
```

Immutable vs Mutable

```
>>> word = "game"
```

```
>>> word[0] = "l"
```

```
TypeError: object does not support item assignment
```

- **Mutable:** Changeable
- **Immutable:** Unchangeable
- String immutability -- Strings are immutable sequences; can't be changed
 - Tuples are immutable too; Lists are mutable!
- But can **create new strings from existing ones** (like through concatenation)

for loops

- Repeats loop body for each element in a sequence
- Ends when it reaches end of the sequence
- **Sequence:** An ordered list of elements
- **Element:** A single item in a sequence
- **Iterate:** To move through a sequence, in order

for letter in “hello world”:

 print(letter)

Lists

- List: A mutable sequence of any type

- Creating List

```
team = []
```

```
team = ["Alberto", "Juan", "Nuria", "Esther"]
```

- len function

```
len(team)
```

- in operator

```
if "Alberto" in team:
```

```
    print("Hi, Alberto, you are part of the team")
```

- Indexing and slicing

```
team[1], team[:3]
```

- Concatenating lists

```
team + ['Oscar', 'Arantxa']
```

Understanding List Mutability

- **Mutable** => Can be modified
- Lists are mutable
 - Elements (or slices) can be added
 - Elements (or slices) can be modified
 - Elements (or slices) can be removed

Assigning New Element Or Slice

```
>>> team.append('Oscar')  
['Alberto', 'Juan', 'Nuria', 'Esther', 'Oscar']
```

```
>>> team[0] = "Arantxa"  
>>> print(team)  
['Arantxa', 'Juan', 'Nuria', 'Esther', 'Oscar']
```

```
>>> team[3:4] = ["Jon"]  
>>> print(team)  
['Arantxa', 'Juan', 'Nuria', 'Jon']
```

Deleting an Element or a Slice

```
>>> team = ['Arantxa', 'Juan', 'Nuria', 'Jon', 'Alberto', 'Esther', 'Oscar']
```

```
>>> del(team[2])
```

```
>>> print(team)
```

```
['Arantxa', 'Juan', 'Jon', 'Alberto', 'Esther', 'Oscar']
```

```
>>> del(team[:2])
```

```
>>> print(team)
```

```
['Jon', 'Alberto', 'Esther', 'Oscar']
```

Strings and lists

```
>>> quote = ' I think there is a world market for maybe  
five computers'
```

```
>>> quote.split()
```

```
['I', 'think', 'there', 'is', 'a', 'world', 'market', 'for', 'maybe',  
'five', 'computers.']
```

```
>>> word = 'engineering'
```

```
>>> list(word)
```

```
['e', 'n', 'g', 'i', 'n', 'e', 'e', 'r', 'i', 'n', 'g']
```


The range() function

```
>>> range(5)
```

```
[0, 1, 2, 3, 4]
```

```
>>> range(0, 50, 5)
```

```
[0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
```

```
>>> range(10,1,-1)
```

```
[10, 9, 8, 7, 6, 5, 4, 3, 2]
```

Returns **a sequence** of integers in range

- `range(i)` returns sequence 0 through $i - 1$
- `range(i, j)` returns sequence i through $j - 1$
- `range(i, j, k)` returns sequence i to $j - 1$, step k

The range() function

counting forward

```
for i in range(10):  
    print(i)
```

counting by fives

```
for i in range(0, 50, 5):  
    print(i)
```

counting backwards

```
for i in range(10, 0, -1):  
    print(i)
```

List comprehension

- `word = 'engineering'`

- `letters = list(word)`

- `[s for s in word]`

`['e', 'n', 'g', 'i', 'n', 'e', 'e', 'r', 'i', 'n', 'g']`

- `>>> [x ** 2 for x in range(20)]`

`[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361]`

List comprehension

- `[x for x in range(20) if x % 2 == 0]`
- `lst_lst = [[1,2,3,4,5], [6,7,8], [9,10]]`
- `lst = [y for x in lst_lst if len(x) < 4 for y in x if y % 2 == 0]`

Lists

- `list.append(x)` - Add an item to the end of the list;
- `list.extend(L)` - Extend the list by appending all the items in the given list;
- `list.insert(i, x)` - Insert an item at a given position. The first argument is the index of the element before which to insert, so `a.insert(0, x)` - inserts at the front of the list, and `a.insert(len(a), x)` is equivalent to `a.append(x)`.
- `list.remove(x)` - Remove the first item from the list whose value is x. It is an error if there is no such item.
- `list.pop([i])` - Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list.
- `list.index(x)` - Return the index in the list of the first item whose value is x
- `list.count(x)` - Return the number of times x appears in the list.
- `list.sort(cmp=None, key=None, reverse=False)` - Sort the items of the list in place
- `list.reverse()` - Reverse the elements of the list, in place.

Tuples

- **Tuple:** Immutable sequence of values of any type
- Could have tuple of integers for a high score list, for example
- Tuples elements don't need to all be of same type

```
a = ("Monday", 3, 4.5)
```

Tuple Basics

- Creating an Empty Tuple

```
team = ()
```

- Treating a Tuple as a Condition

```
if not team:  
    print("You are empty-handed.")
```

- Creating a Tuple with Elements

```
team = ('Jon', 'Alberto', 'Esther', 'Oscar')
```

- Looping through a tuple's elements

```
for person in team:  
    print(person)
```

Tuple Immutability

```
>>> team = ('Jon', 'Alberto', 'Esther', 'Oscar')
```

```
>>> team[0] = "Arantxa"
```

```
TypeError: object doesn't support item assignment
```

- Tuples are immutable
- But can create new tuples from existing ones

Unpacking a sequence

```
>>> name, age = ("Alberto", 42)
>>> print(name)
Alberto
>>> print(age)
42
```

Sequence unpacking: Automatically accessing each element of a sequence as a result of assignment statement

Processing sequence

```
for name in team:
```

```
    print(name)
```

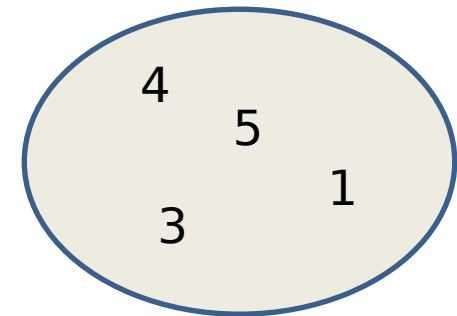
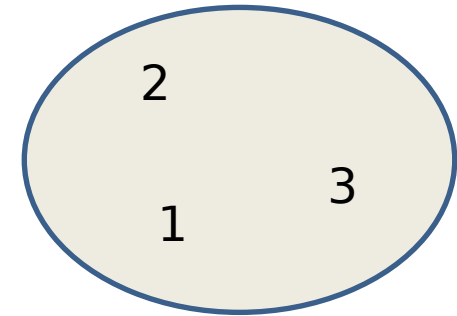
```
for i in range(len(team))
```

```
    print("{}.- {}".format(i, team[i]))
```

```
for i, name in enumerate(team):
```

Sets

- Mathematical set: a collection of values, without duplicates or order
- Order does not matter
 $\{ 1, 2, 3 \} == \{ 3, 2, 1 \}$
- No duplicates
 $\{ 3, 1, 4, 1, 5 \} == \{ 5, 4, 3, 1 \}$
- For every data structure, ask:
 - How to create
 - How to query (look up) and perform other operations
 - (Can result in a new set, or in some other datatype)
 - How to modify



Create a set

1. Direct mathematical syntax

```
odd = { 1, 3, 5 }
```

```
prime = { 2, 3, 5 }
```

Cannot express empty set: “{ }” is a dictionary

2. Construct from a list

```
odd = set([1, 3, 5])
```

```
prime = set([2, 3, 5])
```

```
empty = set([])
```

Python always prints using this syntax

Modifying a set

- Add one element to a set:

```
myset.add(new_ele)
```

```
myset = myset | { new_ele }
```

- Remove one element from a set:

```
myset.remove(elt) # elt must be in myset
```

```
myset.discard(elt) # never errs
```




```
myset = myset - { elt }
```

- Choose and remove some element from a set:

```
myset.pop()
```

Set operations

```
odd = { 1, 3, 5, 7 }  
prime = { 2, 3, 5, 7 }  
even = {2,4,6,8}
```

- membership  Python: `4 in prime` \wedge False
- union  Python: `odd | prime` \wedge { 1, 2, 3, 5 }
- intersection  Python: `odd & prime` \wedge { 3, 5 }
- difference \ or - Python: `odd - prime` \wedge { 1 }
- Iteration over sets:
 # iterates over items in *arbitrary* order
 for item in myset:
 print(item)

Dictionaries or mappings

- A dictionary maps each key to a value
- Order does not matter
- Given a key, can look up a value
 - Given a value, cannot look up its key
- No duplicate keys
 - Two or more keys may map to the same value
- Keys and values are Python values
 - Keys must be immutable (not a list, set, or dict)
- Can add key → value mappings to a dictionary
 - Can also remove (less common)

Dictionary syntax in Python

```
d = { }
```

```
d = dict()
```

```
alberto = {  
    'name': "Alberto",  
    'age': 25,  
    'city': "Pamplona" }
```

```
juan = {  
    'name': "Ana",  
    'age': 28,  
    'city': "Madrid" }
```


Iterating through a dictionary

```
atomic_number = {"H":1, "O":8, "C":6, "Fe":26, "Au":79}
```

```
# Print out all the keys:
```

```
for element_name in atomic_number.keys():  
    print element_name
```

```
# Another way to print out all the keys:
```

```
for element_name in atomic_number:  
    print element_name
```

```
# Print out all the values:
```

```
for element_number in atomic_number.values():  
    print element_number
```

```
# Print out the keys and the values
```

```
for (ele_name, ele_number) in atomic_number.items():  
    print("The atomic number of {} is {}".format(ele_name, ele_number))
```

Modifying a dictionary

```
alberto["City"] = "Barcelona" # change mapping
```

```
alberto["Country"] = "Spain" # change mapping
```

```
del(alberto["City"]) # remove mapping
```

Data types. Sequences

- Mutables:

- list: ordered collection of values of any type which can be added and deleted
 - ['hello', 4, (1, 2), {3, 4}]
- set: unordered, not repeated collection of values of any type which can be added and deleted
 - {'infinite', 1, 0, 5, ('a', 1)}
- dict: unordered collection of key:value pairs. Key can be any immutable type, value can be of any type
 - {'tomato': (1, 'Kg'), 'cucumber': 2, 'salt': '1 spoon', 'aceite': .1}

Data types. Sequences

- Inmutables:
 - str: An ordered sequence of characters
 - “Hello world!”
 - frozenset : A set object is an unordered collection of immutable values
 - `frozenset({3,5,6.1})`
 - tuple: sequence of 0, 1 or n elements (can be of different type)
`(1,'a',3,3)`