Advanced Python Programming

Yan-Fu Kuo
Dept. of Biomechatronics Engineering
National Taiwan University

→

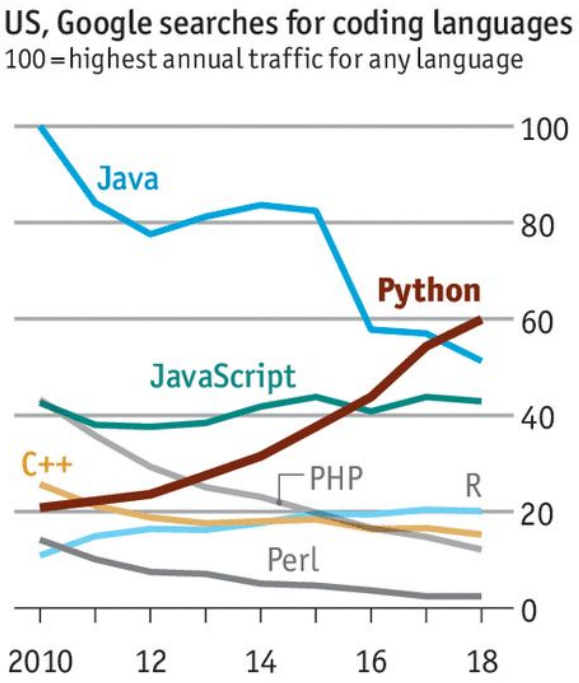
Contents

01 Programming Basics

The Economist

Python is becoming the world's most popular coding language

- CIA has used it for hacking.
- Google for crawling webpages.
- Pixar for producing movies.
- Spotify for recommending songs.
- Some of the most popular packages harness "machine learning".



https://www.economist.com/graphic-detail/2018/07/26/python-is-becoming-the-worlds-most-popular-coding-anguage?fsrc=scn/fb/te/bl/ed/pythonisbecomingtheworldsmostpopularcodinglanguagedailychart

C++ vs. Python

C++

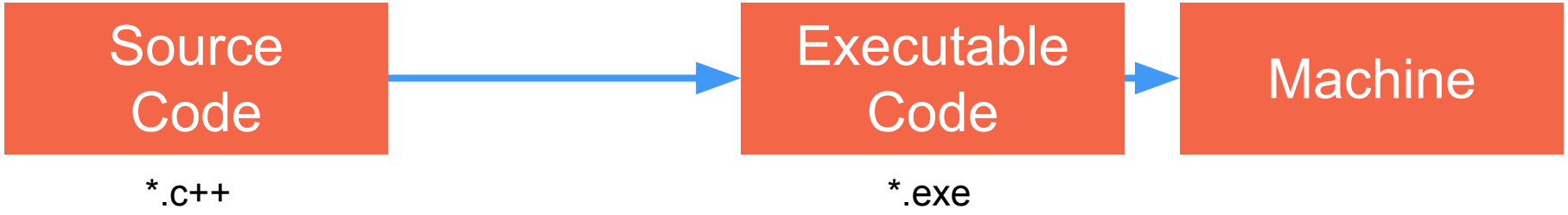
```
#include
#include
using namespace std;
int main() {
string name;
cin >> name;
cout << "Good evening, " << name << endl;
return 0;
}
```

Python

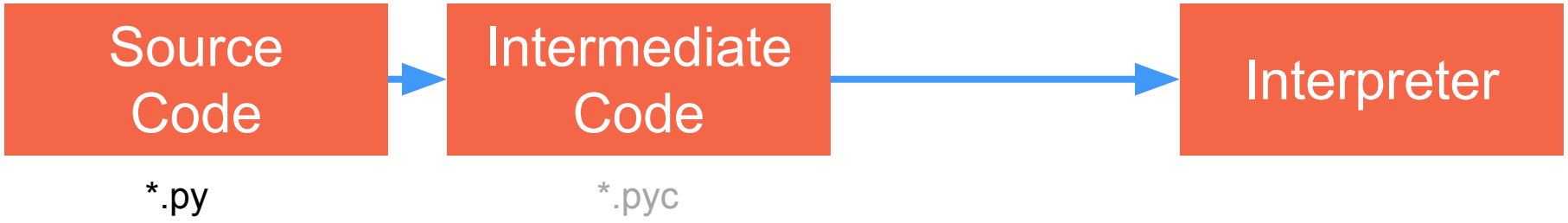
```
name = input()
print("Good evening, " + name)
```

Compiler vs. Interpreter

Compiler (C++)



Interpreter (Python)



Quick Start

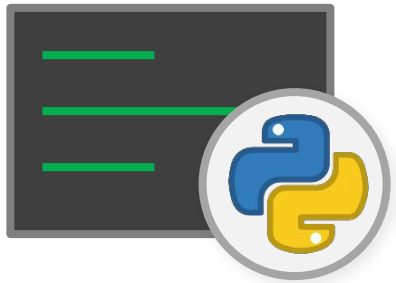
1. Load module with import
2. No curly braces !
3. Comment using #
4. No parentheses needed
5. Boolean operators are words
6. Newline automatically added
7. Extend a statement using \
8. List comprehensions

```
from random import randrange

def numberizer():
    # Generate a random number from 1 to 10.
    return randrange(1, 11)

number = numberizer()

if number>5 and number<=10:
    print("This number is big!") \n

class RandomNumberHolder(object):
    def __init__(self):
        self.numbers = \
        [numberizer(x) for x in range(20)]
```

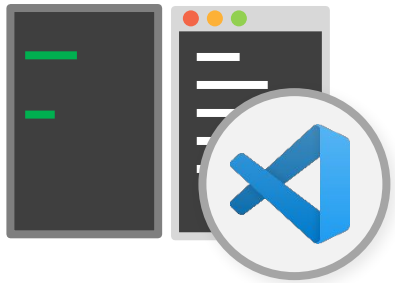
Running Python

The Python interpreter is a program that reads and executes Python code.

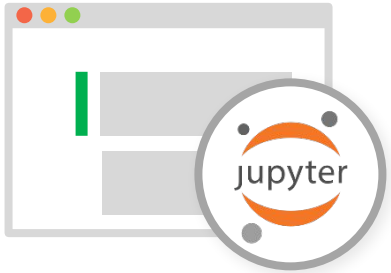
Run Interactively



Run Scripts



Run in Jupyter



Using Python as a Calculator

```
$ python
```

```
$ python3
```

```
Python 3.7.7 (tags/v3.7.7:d7c567b08f, Mar 10 2020, 10:41:24) [MSC
v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Arithmetic operators

Name	Symbol
Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulus	%
Exponentiation	**
Floor Division	//

Bit wise operators (https://wiki.python.org/moin/BitwiseOperators)

Values and Types

```
>>> type(2)
<class 'int'>
>>> type(42.0)
<class 'float'>
>>> type('Hello, World!')
<class 'str'>
```

```
>>> type('2')
<class 'str'>
>>> type('42.0')
<class 'str'>
```

Variables

```
>>> message = 'something different'
>>> n = 17
>>> pi = 3.14159265
```

```
>>> type(message)
<type 'str'>
>>> type(n)
<type 'int'>
>>> type(pi)
<type 'float'>
```

int

n → 17

float

pi → 3.14159265

Variable Names and Keywords

The underscore character, _, can appear in a name. It is often used in names with multiple words, such as `my_name` or `airspeed_of_unladen_swallow`.

```
>>> 76trombones = 'parade'
SyntaxError: invalid syntax
>>> more@ = 1000000
SyntaxError: invalid syntax
>>> class = 'Advanced'
SyntaxError: invalid syntax
```

False	class	finally	is	return
None	in	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	break
except	raise	continue		

String Operations

```
>>> first = 'throat'
>>> second = 'warbler'
>>> first + second
throatwarbler
```

```
>>> 'Spam'*3
SpamSpamSpam
```

Comments

single-line comments

#

multi-line comments

" " "

" " "

```
"""Example with types documented in the docstring."""
import cv2
import numpy as np

def chlorophyll(fname):
  """ Get chlorophyll index (1-6) of a image """

  img = vari(fname)
  one = (img >= 100)
  two = (img >= 90) & (img < 100)
  three = (img >= 65) & (img < 90)
  # four = (img >= 45) & (img < 65)
  # five = (img >= 20) & (img < 45)
  # six = (img > 0) & (img < 20)

  return img
```

Python docstring

Quiz 1

1. In a print statement, what happens if you leave out one of the parentheses, or both?
2. You can use a minus sign to make a negative number like -2. What happens if you put a plus sign before a number? What about 2++2?
3. In math notation, leading zeros are ok, as in 09. What happens if you try this in Python? What about 011?
4. In some languages every statement ends with a semi-colon, ;. What happens if you put a semi-colon at the end of a Python statement?

Functions

y = function(x)

```
>>> type(42)
<class 'int'>
>>> int('32')
32
>>> int('Hello')
ValueError: invalid literal for int():
Hello
>>> int(3.99999)
```

Why Functions?

- To make your program easier to read and debug
- To make a program smaller by eliminating repetitive code
- Dividing a long program into functions allows you to debug the parts one at a time and then assemble them into a working whole.
- Well-designed functions are often useful for many programs. Once you write and debug one, you can reuse it.

Built-in Math Functions

```
>>> import math

>>> math
<module 'math' (built-in)>

>>> ratio = 0.8
>>> decibels = 10 * math.log10(ratio)

>>> radians = 0.7
>>> height = math.sin(radians)
```

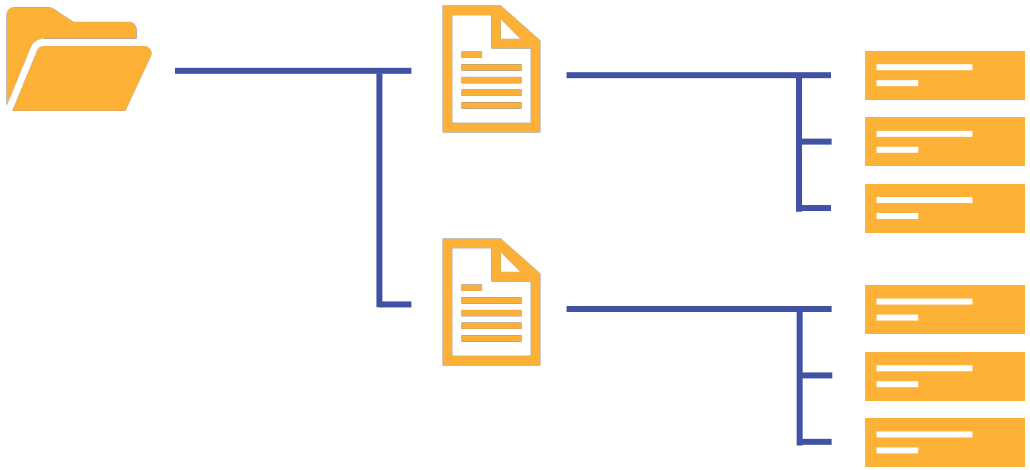
dot notation

Python Package

Package
(Library)

Module

Functions



https://www.python.org/dev/peps/pep-0328/#guido-s-decision

Self-defined Functions

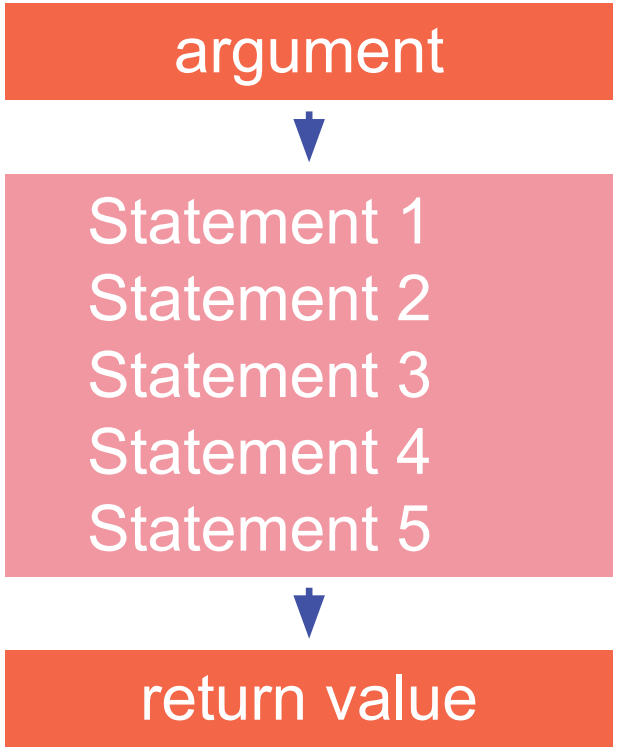
A function definition specifies the name of a new function and the sequence of statements that run when the function is called.

```
print_lyrics()
```

```
>>> def print_lyrics():
...     print("I'm a lumberjack")
...     print("and I’m okay.")
...
```

Parameters and Arguments

Inside the function, the **arguments** are assigned to variables called **parameters**.



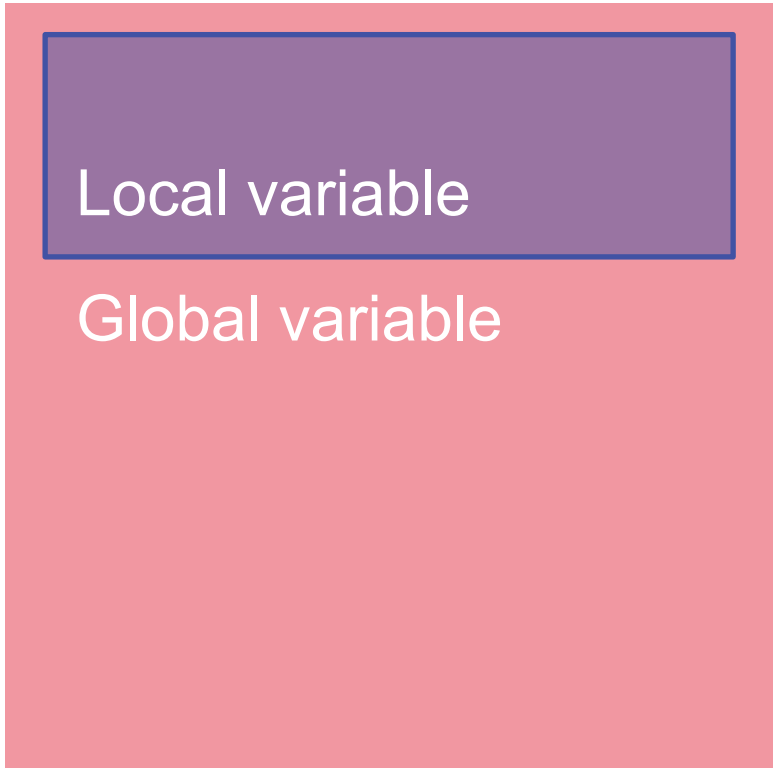

```
>>> def print_twice(whatever):
...     print(whatever)
...     print(whatever)
...
```

whatever = 42

Variables & Parameters are Local

```
def cat_twice(part1, part2):
    cat = part1 + part2
    print_twice(cat)
```

```
>>> cat_twice('Bing ', 'tiddle.')
Bing tiddle.
Bing tiddle.
>>> print(cat)
NameError: name 'cat' is not defined
```



Quiz 2

Question 1

```
>>> def f():
...     print(s)
>>> s = "AAAA"
>>> f()
```

AAAA

Question 2

```
>>> def f():
...     s = "AAAA"
...     print(s)
>>> s = "BBBB"
>>> f()
>>> print(s)
```

AAAA
BBBB

Question 3

```
>>> def f():
...     print(s)
...     s = "AAAA"
...     print(s)
>>> s = 'BBBB'
>>> f()
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in f
UnboundLocalError: local variable 's'
referenced before assignment
```

The scope of index variables in Python's for loops

Return Values

```
>>> radians = 0.7
>>> height = math.sin(radians)
```

```
def absolute_value(x):
    if x < 0:
        return -x
    if x > 0:
        return x
```

```
def area(radius):
    a = math.pi * radius**2
    return a
```

```
>>> print(absolute_value(0))
None
```

Y.F. Kuo & K.T. Yeh

02 Python Syntax & Datatypes

Boolean Expressions

A **boolean expression** is an expression that is either true or false.

```
>>> 5 == 5
True
>>> 5 == 6
False
```

```
x != y
x > y
x < y
x >= y
x <= y
```

Logical Operators

- AND
- OR
- NOT

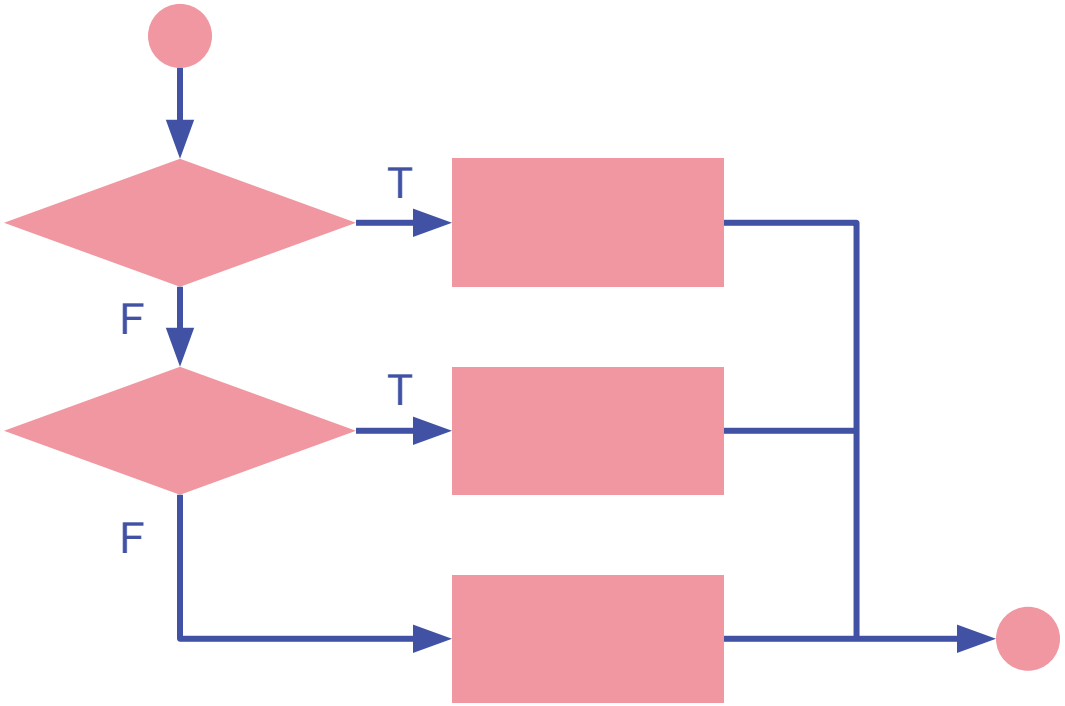
```
and
```

```
or
```

```
not
```

Conditional Execution

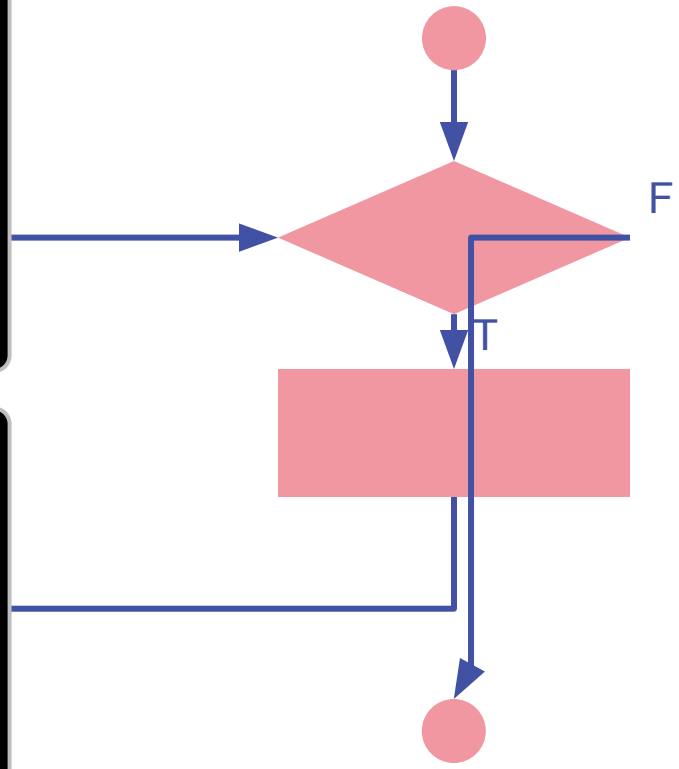
```
if condition1:
    statement1
elif condition2:
    statement2
else:
    statement3
```



The while Statement

```
def countdown(n):
  while n > 0:
    print(n)
    n = n - 1
  print('Blastoff!')
```

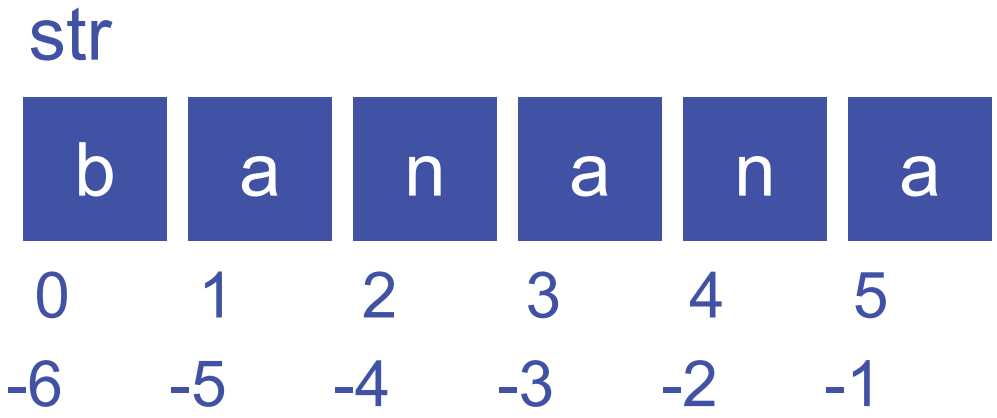
```
while True:
  line = input('> ')
  if line == 'done':
    break
  print(line)
```



String

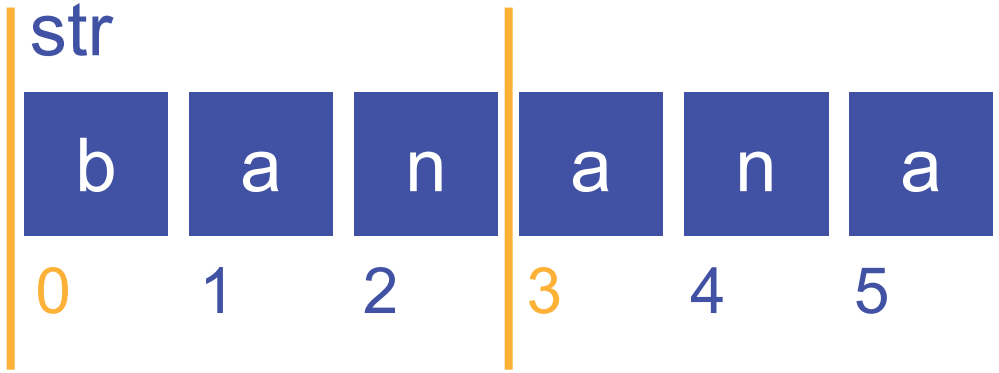
A String is a Sequence

```
>>> fruit = 'banana'
>>> fruit[1]
'a'
>>> letter = fruit[0]
>>> letter
'b'
>>> len(fruit)
6
```



String Slices

```
>>> fruit = 'banana'
>>> fruit[0:3]
'ban'
>>> fruit[3:]
'ana'
>>> fruit[3:3]
''
```



What do you think fruit[:] means?

Strings are Immutable

```
>>> greeting = 'Hello, world!'
>>> greeting[0] = 'J'
TypeError: 'str' object does not
support item assignment
```

```
>>> greeting = 'Hello, world!'
>>> new_greeting = 'J' + greeting[1:]
>>> new_greeting
'Jello, world!'
```

Quiz 3 Traversal with a for Loop

while loop

```
index = 0
while index < len(fruit):
    letter = fruit[index]
    print(letter)
    index = index + 1
```

for loop

```
for letter in fruit:
    print(letter)
```

List

A list is a sequence of values

```
[10, 20, 30, 40]
['crunchy frog', 'ram bladder', 'lark vomit']
['spam', 2.0, 5, [10, 20]]
[]
```

Not like strings, lists are <u>mutable</u>.

```
>>> numbers = [42, 123]
>>> numbers[1] = 5
>>> numbers
[42, 5]
```

List Operations

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> c
[1, 2, 3, 4, 5, 6]
```

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

List Slicing

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1:3]
['b', 'c']
>>> t[:4]
['a', 'b', 'c', 'd']
>>> t[3:]
['d', 'e', 'f']
```

```
>>> t[1:3] = ['x', 'y']
>>> t
['a', 'x', 'y', 'd', 'e', 'f']
```

List Methods

```
>>> t = ['a', 'b', 'c']
>>> t.append('d')
>>> t
['a', 'b', 'c', 'd']
```

```
>>> t1 = ['a', 'b', 'c']
>>> t2 = ['d', 'e']
>>> t1.extend(t2)
>>> t1
['a', 'b', 'c', 'd', 'e']
```

Try `t1.append(t2)`

Other methods:

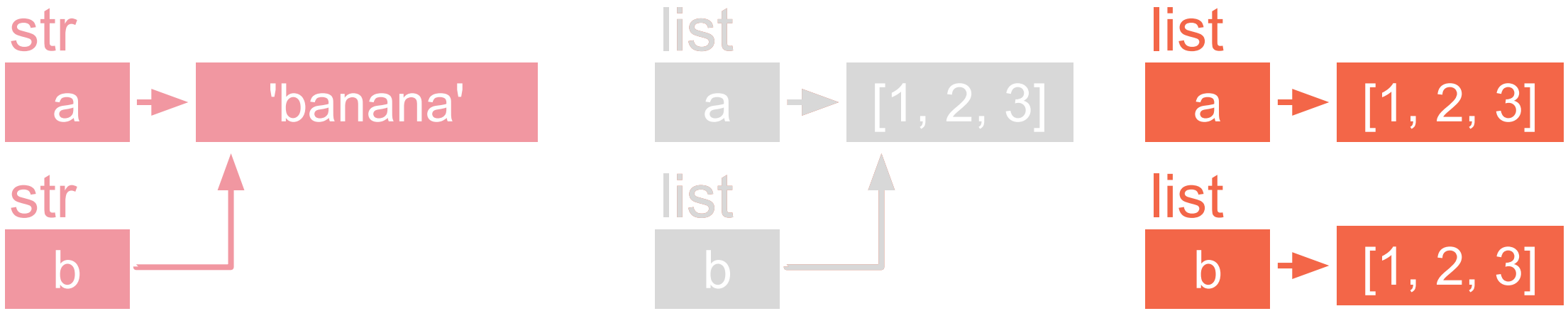
Clear()	Pop()
Copy()	Remove()
Count()	Reverse()
Index()	Sort()
Insert()	

https://docs.python.org/3/tutorial/datastructures.html

Objects and Values

```
>>> a = 'banana'
>>> b = 'banana'
>>> a is b
True
```

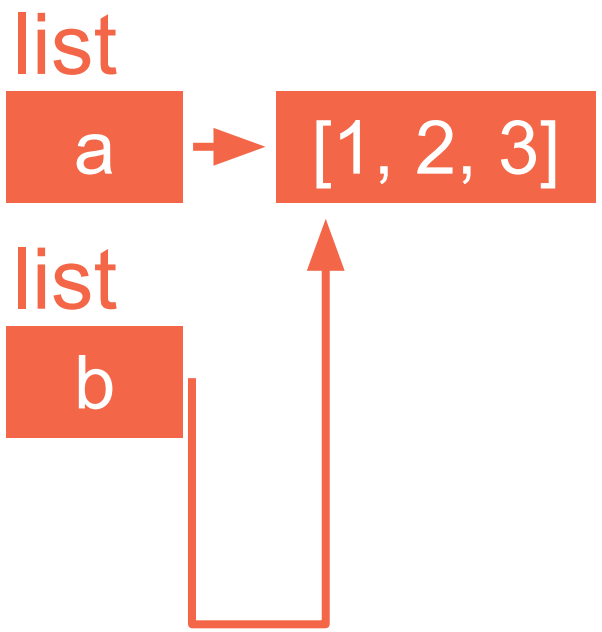
```
>>> a = [1, 2, 3]
>>> b = [1, 2, 3]
>>> a is b
False
```



Aliasing

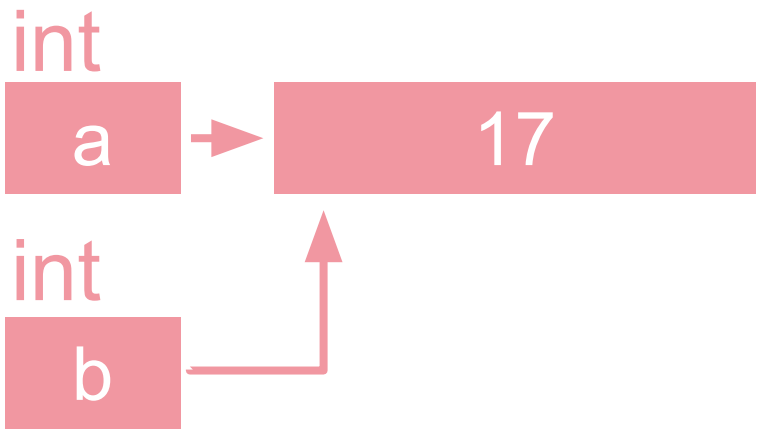
```
>>> a = [1, 2, 3]
>>> b = a
>>> b is a
True
```

```
>>> b[0] = 42
>>> a
[42, 2, 3]
```



Some Integers Are Pre-allocated

```
>>> a = 17
>>> b = 17
>>> a is b
True
```



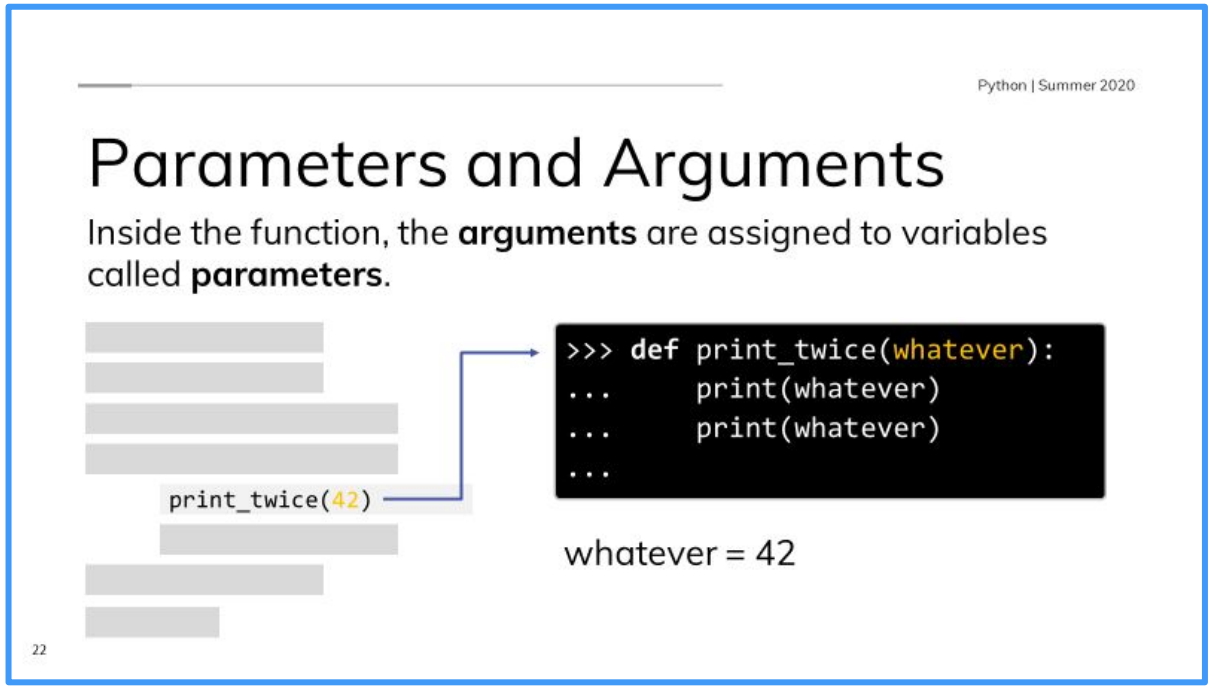
https://www.codementor.io/@arpitbhayani/python-caches-integers-16jih595jk

List as Arguments

```
def delete_head(t):
    del t[0]
```

```
>>> letters = ['a', 'b',
'c']
>>> delete_head(letters)
>>> letters
```

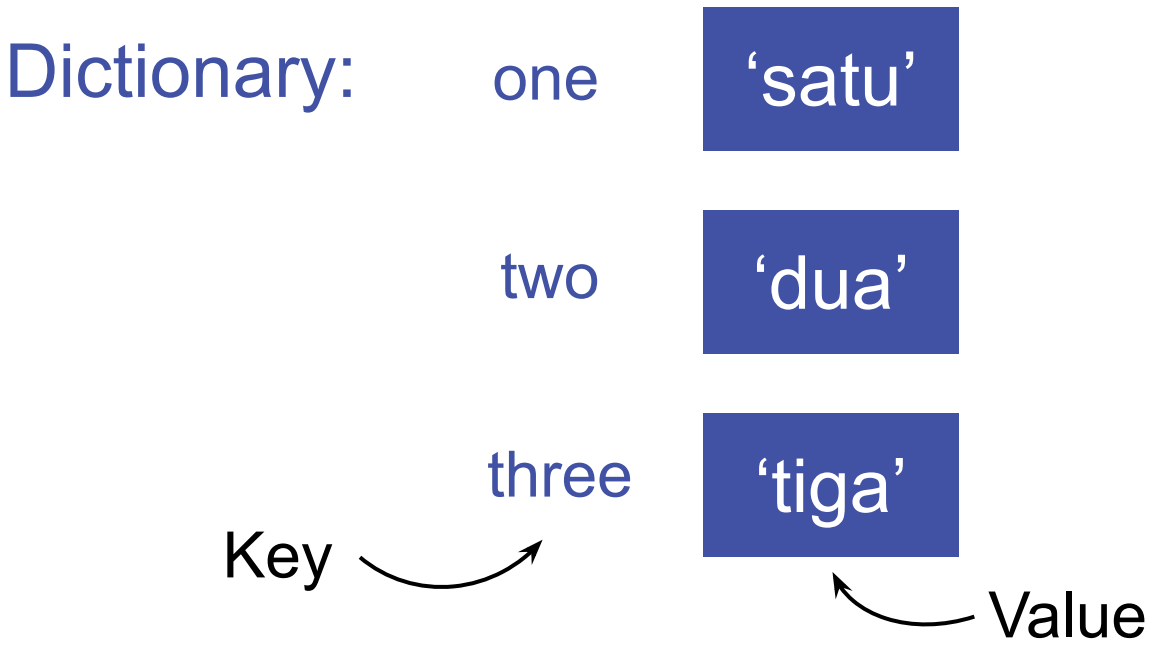
Call by reference



What about String as Arguments?

Dictionary

```
>>> en2in = dict()
>>> en2in
{}
>>> en2in['one'] = 'satu'
>>> en2in
{'one': 'satu'}
```




```
>>> eng2in = {'one': 'satu', 'two': 'dua', 'three': 'tiga'}
>>> eng2in
{'one': 'satu', 'three': 'tiga, 'two': 'dua'}
```

Dictionary Methods

clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Tuples

A **tuple** is a comma-separated sequence of value

```
>>> t = 'a', 'b', 'c', 'd', 'e'
>>> t = ('a', 'b', 'c', 'd', 'e')
```

```
>>> t1 = 'a',
>>> type(t1)
<class 'tuple'>
```

```
>>> t2 = ('a')
>>> type(t2)
<class 'str'>
```

```
>>> t = tuple()
>>> t
()
```

```
>>> t = tuple('lupins')
>>> t
('l', 'u', 'p', 'i', 'n',
```

```
>>> t = ('a', 'b', 'c', 'd',
'e')
>>> t[0]
```

Tuple Assignment

Tuple packing, the values on the left are ‘packed’ together in a tuple

```
>>> julia = ("Julia", "Roberts", 1967,
"Duplicity", 2009, "Actress", "Atlanta, Georgia")
```

Tuple unpacking, the values in a tuple on the right are ‘unpacked’ into the variables/names on the right

```
>>> (name, surname, b_year, movie, m_year,
profession, b_place) = julia
>>> name
"Julia"
```

Tuples as Return Values

```
def f(r):
    """ Return (circumference, area) of a circle of radius r """
    c = 2 * math.pi * r
    a = math.pi * r * r
    return (c, a)
```

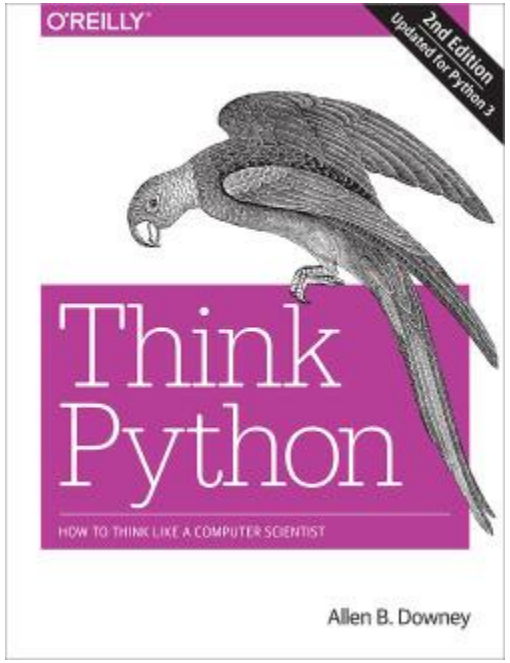
```
>>> circumference, area = f(20)
```

Reference

Downey, A. B., Brooks Jr, F. P., Peek, J., Todino, G., Strang, J., Robbins, A., ... & Cameron, D. (2012). Think Python 2e. Green Tea Press Supplemental Material:.

http://greenteapress.com/thinkpython2/html/index.html

https://github.com/AllenDowney/ThinkPython2



thank you!