DA1: DESIGNING APPLICATIONS IN PYTHON

Aidos Sarsembayev

the lectures content is taken from Stanford's course CS41

THE PURPOSE OF THE COURSE

- To teach you the basics…
- Beyond the basics…
- Data structures, functions
- Functional / OO programming
- The Python Standard Library
- Third-party tools/libs
- Build ~10 apps (console, desktop, web)

A BIT OF HISTORY

- Guido van Rossum – the author
- The development started in the late 1980
- The first release in 1991
- Initially was an OOP oriented language

the picture is takes from Wikipedia

THE PHILOSOPHY OF PYTHON (1)

The Zen of Python, by Tim Peters

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.

THE PHILOSOPHY OF PYTHON (2)

- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one-- and preferably only one --obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea -- let's do more of those!

Programmers are more important than programs

“HELLO WORLD” IN JAVA

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

“HELLO WORLD” IN C++

```
#include <iostream>
using namespace std;

int main() {
		cout << "Hello World!" << endl;
}
```

“HELLO WORLD” IN PYTHON

```
print("Hello world!")
```

PYTHON IN BUSINESS

Dropbox

Instagram

Google

YouTube

Quora

OTHER PYTHON USERS

INSTALLATION

- Go to https://www.python.org/
- Current version is 3.7.0

PYTHON 2.7 VS 3.6

- Python 2.7 is slowly dying.
- We’re gonna be using Python 3

WHAT KINDS OF PYTHON PROGRAMS EXIST?

- Console
- Desktop
- Web

WHAT KIND OF IDES ARE THERE?

- Atom
- Brackets
- Jupyter
- PyCharm
- Spyder
- etc.

PYTHON BASICS

- Interactive Interpreter
- Comments
- Variables and Types
- Numbers and Booleans
- Strings and Lists
- Console I/O
- Control Flow
- Loops
- Functions

INTERACTIVE INTERPRETER

```
C:\Users\Aidos> python

Python 3.6.4 (v3.4.3:9b73f1c3e601, Jan 16 2018, 02:52:03)
[GCC 4.2.1 (AMD 64) on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

You can start writing your code right here…

THE ADVANTAGES OF INTERACTIVE INTERPRETER

- Immediate gratification!
- Sandboxed environment to experiment with Python
- Shortens code-test-debug cycle to seconds

COMMENTS

#

# Single line comments start with a '#‘

MULTILINE COMMENTS

```
"""
Multiline strings can be written
using three "s, and are often used
as function and module comments
"""
```

VARIABLES

- `x = 2              # No semicolon!`

VARIABLES

- x = 2   # No semicolon!
- x * 7
- # => 14

- x = "Hello, I'm “   # now it’s a String

VARIABLES

- x = 2 # No semicolon!
- x * 7
- # => 14

- x = "Hello, I'm “ # now it’s a String
- x + "Python!"
- # => "Hello, I'm Python“ # String concat.

In Java or C++

- int x = 0;

In Python

- x = 0;

Variables in Python are dynamically-typed: declared without an explicit type

However, objects have a type, so Python knows the type of a variable, even if you don't

Variables in Python are dynamically-typed: declared without an explicit type

However, objects have a type, so Python knows the type of a variable, even if you don't

- type(1) # => <class 'int'>
- type("Hello") # => <class 'str'>
- type(None) # => <class 'NoneType'>

Variables in Python are dynamically-typed: declared without an explicit type

However, objects have a type, so Python knows the type of a variable, even if you don't

- type(1) # => <class 'int'>
- type("Hello") # => <class 'str'>
- type(None) # => <class 'NoneType'>

- type(int) # => <class 'type'>
- type(type(int)) # => <class 'type'>

NUMBERS AND MATH

- 3 # => 3 (int)
- 3.0 # => 3.0 (float)

Python has two numeric types int and float

- 1 + 1 # => 2
- 8 – 1 # => 7
- 10 * 2 # => 20
- 9 / 3 # => 3.0
- 5 / 2 # => 2.5
- 7 / 1.4 # => 5.0

NUMBERS AND MATH

Python has two numeric types int and float

- 1 + 1 # => 2
- 8 - 1 # => 7
- 10 * 2 # => 20
- 9 / 3 # => 3.0
- 5 / 2 # => 2.5
- 7 / 1.4 # => 5.0

- 7 // 3 # => 2 (integer division)
- 7 % 3 # => 1 (integer modulus)

BOOLEANS

bool is a subtype of int, where True == 1 and False == 0

- True # => True
- False # => False
- not True # => False
- True and False # => False
- True or False # => True (short-circuits)
- 1 == 1 # => True
- 2 * 3 == 5 # => False
- 1 != 1 # => False
- 2 * 3 != 5 # => True

BOOLEANS

- True # => True
- False # => False
- not True # => False
- True and False # => False
- True or False # => True (short-circuits)
- 1 == 1 # => True
- 2 * 3 == 5 # => False
- 1 != 1 # => False
- 2 * 3 != 5 # => True

BOOLEANS

- True # => True
- False # => False
- not True # => False
- True and False # => False
- True or False # => True (short-circuits)
- 1 == 1 # => True
- 2 * 3 == 5 # => False
- 1 != 1 # => False
- 2 * 3 != 5 # => True
- 2 >= 0 # => True
- 1 < 2 < 3 # => True (1 < 2 and 2 < 3)
- 1 < 2 >= 3 # => False (1 < 2 and 2 >= 3)

STRINGS

- No char in Python!
- Both ' and " create string literals

STRINGS

- No char in Python!
- Both ' and " create string literals

- greeting = 'Hello'
- group = "wørld" # Unicode by default
- greeting + ' ' + group + '!' # => 'Hello wørld!'

INDEXING

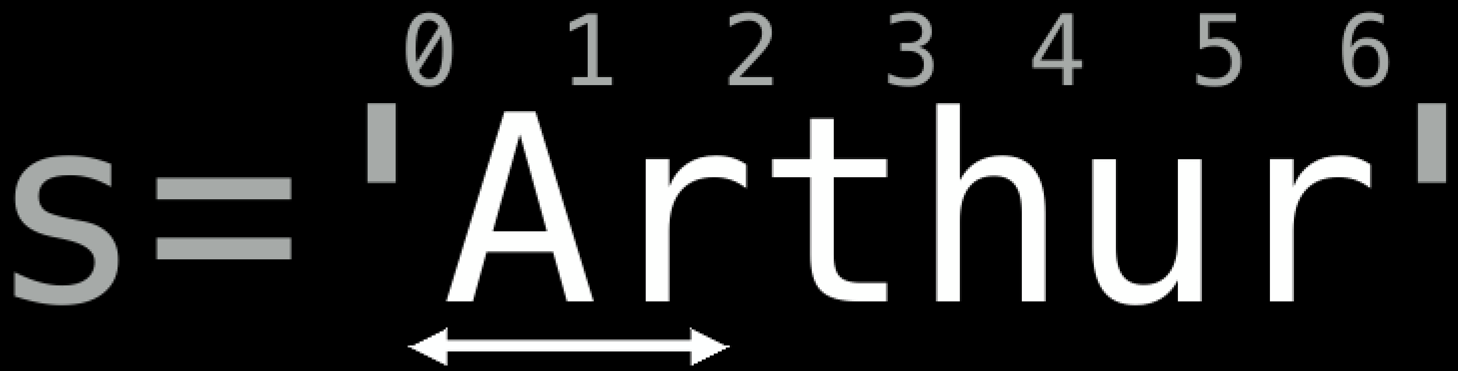
Starts from 0 as usual

- 0 1 2 3 4 5
- s='Arthur‘

There is also negative indexing

- s='-5:A -4:r -3:t -2:h -1:u 0:r'

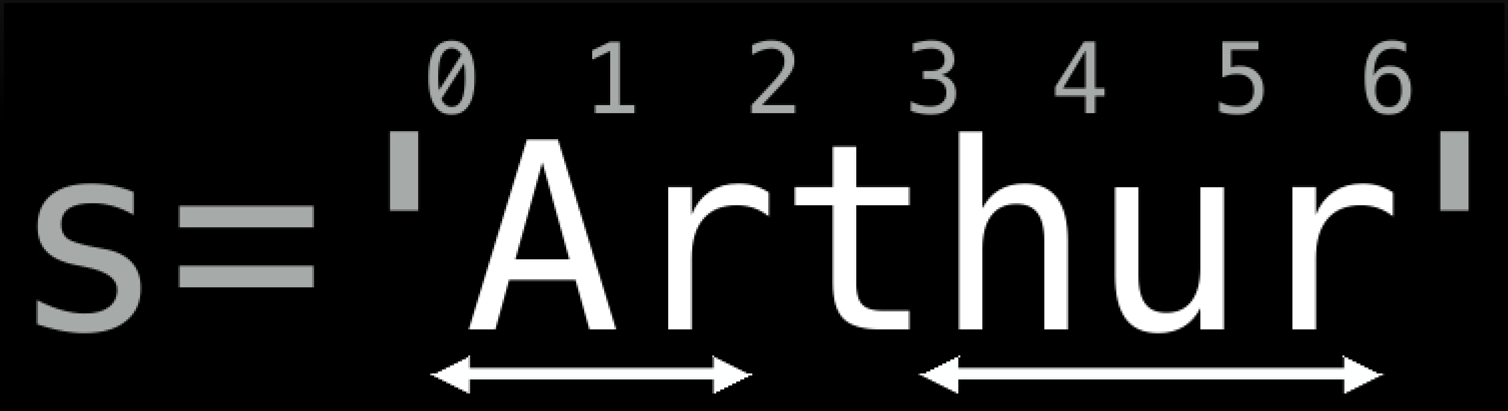
SLICING



```
s='Arthur'
```

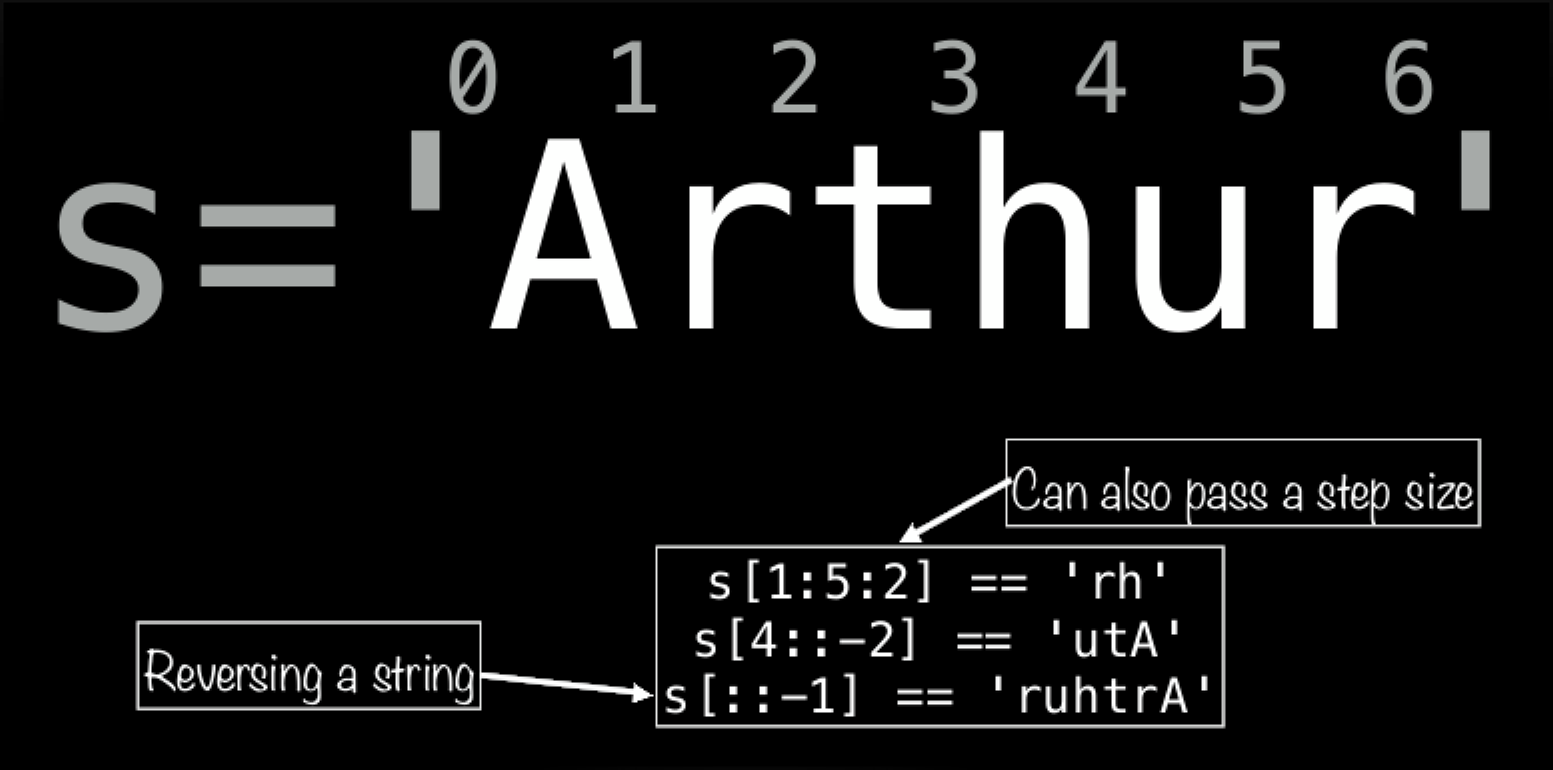
```
s[0:2] ==  'Ar'
```

SLICING



```
s[0:2] ==  'Ar'
s[3:6] == 'hur'
```

SLICING



```
s[1:5:2] == 'rh'
s[4::-2] == 'utA'
s[::-1] == 'ruhtrA'
```

CONVERTING VALUES

- str(42) # => "42"
- int("42") # => 42
- float("2.5") # => 2.5
- float("1") # => 1.0

LISTS

Here is a list

- easy_as = [1,2,3]

- Square brackets delimit lists
- Commas separate elements

BASIC LISTS

# Create a new list

- empty = []
- letters = ['a', 'b', 'c', 'd']
- numbers = [2, 3, 5]

BASIC LISTS

# Create a new list

- empty = []
- letters = ['a', 'b', 'c', 'd']
- numbers = [2, 3, 5]

# Lists can contain elements of different types

- mixed = [4, 5, "seconds"]

BASIC LISTS

# Create a new list

- empty = []
- letters = ['a', 'b', 'c', 'd']
- numbers = [2, 3, 5]

# Lists can contain elements of different types

- mixed = [4, 5, "seconds"]

# Append elements to the end of a list

- numbers.append(7) # numbers == [2, 3, 5, 7]
- numbers.append(11) # numbers == [2, 3, 5, 7, 11]

INSPECTING LIST ELEMENTS

# Access elements at a particular index

- numbers[0] # => 2
- numbers[-1] # => 11

INSPECTING LIST ELEMENTS

# Access elements at a particular index

- numbers[0] # => 2
- numbers[-1] # => 11

# You can also slice lists – the same rules apply

- letters[:3] # => ['a', 'b', 'c']

NESTED (ВЛОЖЕННЫЕ) LISTS

# Lists really can contain anything – even other lists!

- x = [letters, numbers]
- x # => [['a', 'b', 'c', 'd'], [2, 3, 5, 7, 11]]

NESTED (ВЛОЖЕННЫЕ) LISTS

# Lists really can contain anything – even other lists!

- x = [letters, numbers]
- x # => [['a', 'b', 'c', 'd'], [2, 3, 5, 7, 11]]
- x[0] # => ['a', 'b', 'c', 'd']
- x[0][1] # => 'b'
- x[1][2:] # => [5, 7, 11]

GENERAL QUERIES

# Length (len)

- len([]) # => 0
- len("python") # => 6
- len([4,5,"seconds"]) # => 3

GENERAL QUERIES

# Length (len)

- len([]) # => 0
- len("python") # => 6
- len([4,5,"seconds"]) # => 3

# Membership (in)

- 0 in [] # => False
- 'y' in 'python' # => True
- 'minutes' in [4, 5, 'seconds'] # => False

CONSOLE I/O

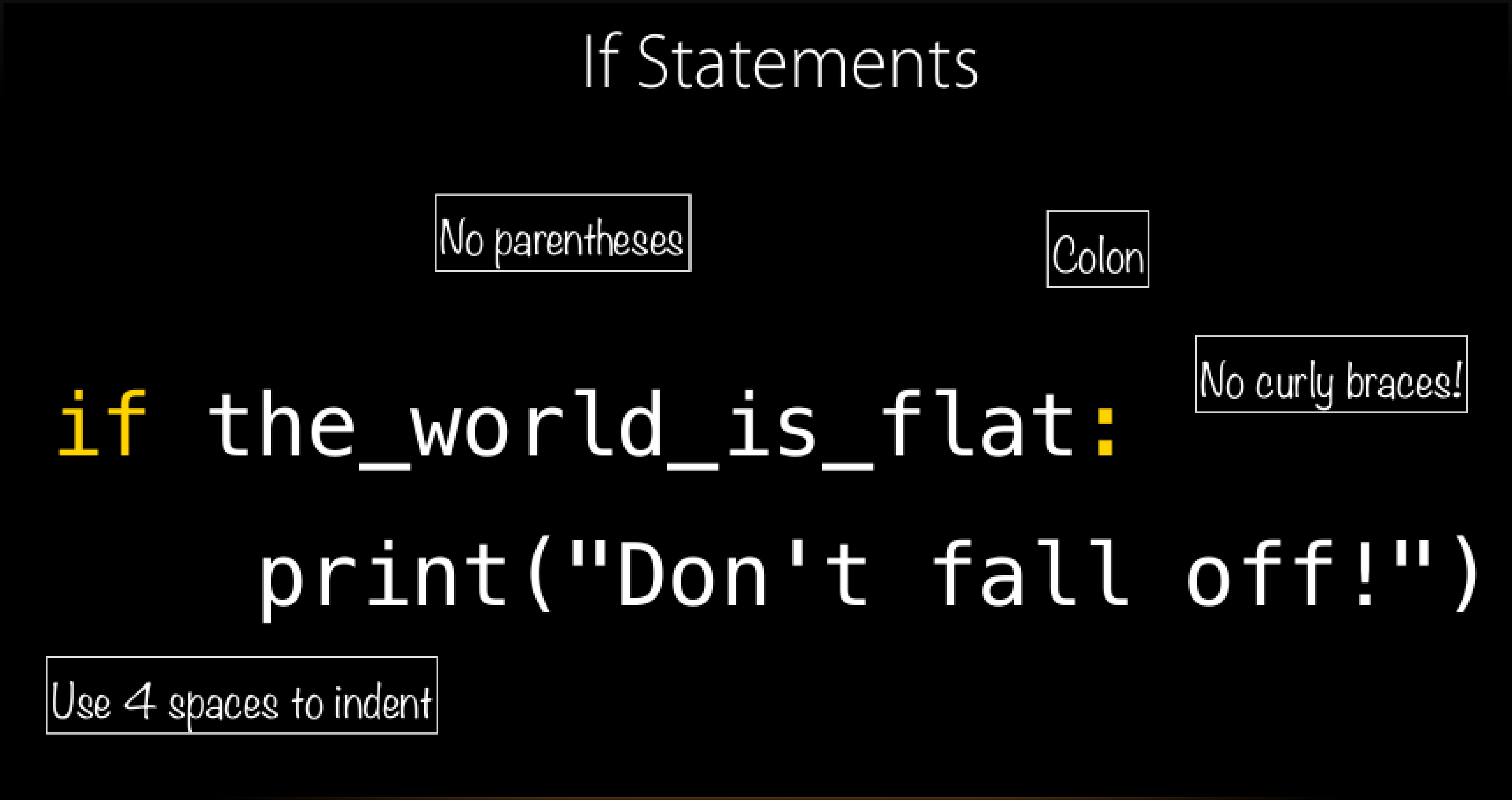
# Read a string from the user

- >>> name = input("What is your name? ")

- >>> print("I'm Python. Nice to meet you,", name)

I'm Python. Nice to meet you, Sam

CONTROL FLOW


If Statements

No parentheses

Colon

No curly braces!

```
if the_world_is_flat:
    print("Don't fall off!")
```

Use 4 spaces to indent

4 SPACES?!

Readability counts

Removes visually-cluttering punctuation

Editor Settings

```
{
	"tab_size": 4,
	"translate_tabs_to_spaces": true,
}
```

IF STATEMENTS

```
if some_condition:
    print('Some condition holds')
elif other_condition:
    print('Other condition holds')
else:
    print('Neither condition holds')
```

- else is optional
- An Aside Python has no switch statement, opting for if/elif/else chains

PALINDROME?

```
# Palindrome Check

word = input("Please enter a word: ")
reversed_word = word[::-1]
if word == reversed_word:
        print("Hooray! You entered a palindrome")
else:
        print("You did not enter a palindrome")
```

TRUTHY AND FALSY

```
# 'Falsy'
•    bool(None)
•    bool(False)
•    bool(0)
•    bool(0.0)
•    bool('')
# Empty data structures are 'falsy'
•    bool([]) # => False
# Everything else is 'truthy‘
# How should we check for an empty list?
data = []
if data:
        process(data):
else:
        print("There's no data!")
```

LOOPS

```
for item in iterable:
        process(item)
```

Loop explicitly over data (for …here… in)

Strings, lists, etc. (iterable)

No loop counter!

RANGE

Used to iterate over a sequence of numbers

- range(3)

# generates 0, 1, 2

- range(5, 10)

# generates 5, 6, 7, 8, 9

- range(2, 12, 3)

# generates 2, 5, 8, 11

- range(-7, -30, -5)

# generates -7, -12, -17, -22, -27

range(stop) or range(start, stop[, step])

BREAK AND CONTINUE

```
for n in range(10):
        if n == 6:
                break
        print(n, end=',')
# => 0, 1, 2, 3, 4, 5,
for n in range(10):
        if n % 2 == 0:
                print("Even", n)
                continue
        print("Odd", n)
```

break breaks out of the smallest enclosing *for* or *while* loop

continue continues with the next iteration of the loop

FUNCTIONS

```
def fn_name(param1, param2):
        value = do_something()
        return value
```

- The **def** keyword defines a function
- Parameters have no explicit types
- return is optional if either return or its value are omitted, implicitly returns None