Crash Course: Python

Khang Vinh | UCLA Computer Science

WHS Site | May 2019

What is Python?

- General purpose language
- How is it used?
  - Backend web development
  - Data analysis
  - Artificial intelligence
  - Productivity tools
  - Scientific computing

KEY TRAITS

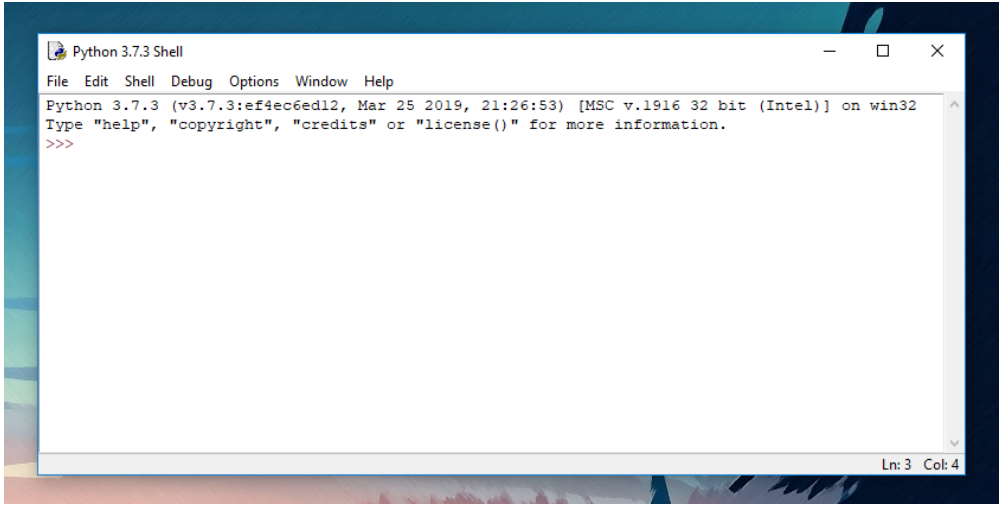
- High-level language
  - Reads like English!
- Dynamically typed
  - Forgiving of errors!
- Interpreter
  - Executed line-by-line

Python is meant to be easy to understand and flexible for the user!

Using Python

- Download/install Python 3.7 executable
- https://www.python.org/

- Welcome to the shell!



Python is the 4th most popular language on GitHub according to the TIOBE index.

The Shell

- Command-line interface
- Stands between user and OS
  - Protection for both parties
- Basic shell usage
  - printing values
  - echoing values
  - simple calculator

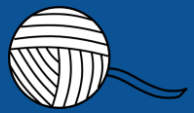
```
>>> print(“Hello World”)
Hello World
>>> “Hello World”
‘Hello World’
>>> 3
3
>>> 4 + 5 * 3 / 2
11.5
>>>
```

Python is, in fact, not named after the snake, but after the comedy sketch group Monty Python!

Basic Data Types

- Integer
  - pos/neg whole numbers
- Float
  - decimal numbers
- String (str)
  - collection of characters
  - String concatenation

```
>>>  ‘Hello’ + ‘World’
‘HelloWorld’
>>>  ‘Hello’ + ‘ ’ + ‘World’
‘Hello World’
>>>  ‘Hello’ + 42
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in
<module>
    'Hello' + 42
TypeError: can only concatenate str
(not "int") to str
>>>
```



Strings must enclosed in single quotes characters!

Variables

- Assignment
  - storing values in boxes
- Choosing variable names
  - python is case sensitive
  - camelcase notation is preferred

Valid	Invalid
currentBalance	current-balance
current_balance	current balance
yeet	4head
YEET	42
pho87	$blessed

Be specific with name variables! Make your life easier!

First Program

- File editor
  - File > New File
  - Type in this program
  - File > Save As > Save
    - (hello.py)
  - Run > Run Module

```
# this program says hello!

print('Hello World!')

print('What is your name?') # ask for name
myName = input()
print('It is good to meet you, ' + myName)
print('The length of your name is:')
print(len(myName))

print('What is your age?') # ask for age
myAge = input()
print('You will be ' + str(int(myAge) + 1) + ' in a year.')
```

Comments are ignored

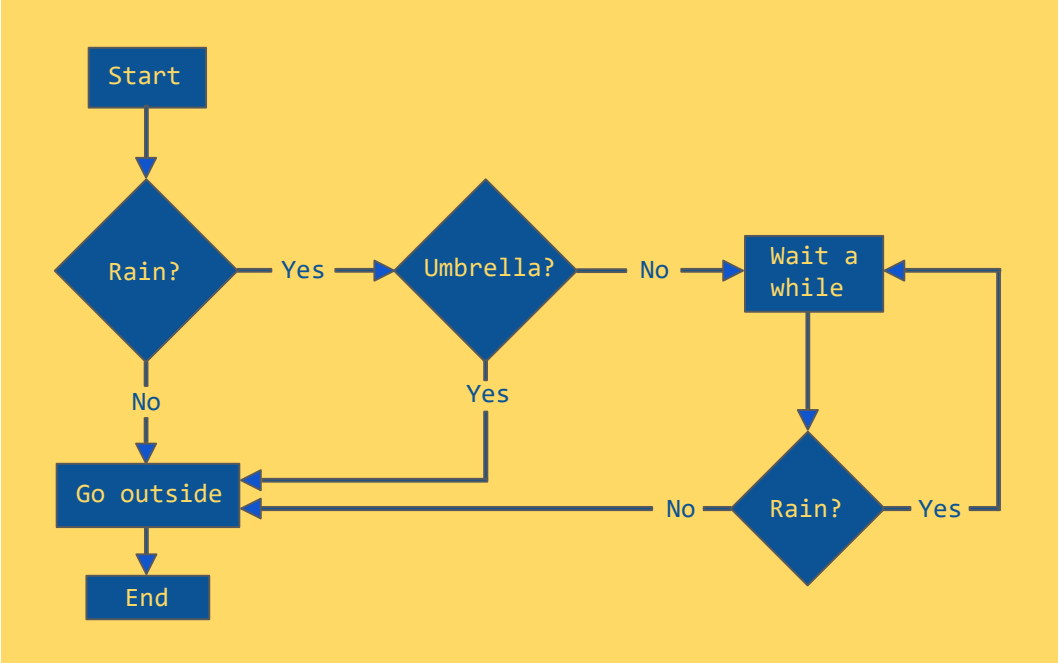
input() is passed a str from user

len() returns the length of str

Don't rush through typing your code. Mistakes will happen!

Control Flow

- Program is a series of instructions
- Real power comes from:
  - iteration
  - skipping
  - selection
- Control flow
  - execution of statements under given conditions
  - flow chart



Flow charts can help visualize a problem more clearly.

Boolean Values

- Data type with 2 values
  - True
  - False

- Can be used in expressed and can be stored in variables

```
>>> soup = True
>>> soup
True
>>> True = 2 + 2
SyntaxError: assignment to keyword
>>>
```

True and False are reserved keywords in Python.

Comparison Operators

- Compares two values; outputs True or False

Operator	Meaning
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

```
>>> 42 == 42
True
>>> ‘hello’ == ‘Hello’
False
>>> 42 <= 100
True
>>> True != False
True
>>> 42 == ‘42’
False
>>>
```

Don’t confuse the assignment operator (=) with the comparison operator (==).

Boolean Operators

- Compares two Boolean values
  - and
  - or
  - not
- Always evaluates to a single Boolean value

AND Operator

Expression	Output
True and True	True
True and False	False
False and True	False
False and False	False

OR Operator

Expression	Output
True and True	True
True and False	True
False and True	True
False and False	False

NOT Operator

Expression	Output
not True	False
not False	True

The basic Boolean operators can be used to create more advanced ones such as XOR, NAND, and NOR.

Boolean Expressions

- Combine comparison operators with Boolean operators
  - Comparison ops evaluate to Boolean
  - Boolean ops operate on Boolean values

```
>>> (4 < 5) and (5 < 6)
True
>>> (4 < 5) and (9 <6)
False
>>> (1 == 2) or (2 == 2)
True
```

```
>>> 2 + 2 == 4 and not 2 + 2 == 5 and 2 * 2 == 2 + 2
True
```

When evaluating Boolean expressions, it helps to add parentheses.

Elements of Control Flow

- Conditions
  - Boolean values in the context of control flow
- Blocks of code
  - Lines of Python code can be grouped in blocks
- Three rules
  - Blocks begin when indentation increases
  - Blocks can contain sub-blocks
  - Blocks end when indentation decreases to zero or to a containing block's indentation

```
# example of blocks

if name == 'Joe':
            print('Hello, Joe')
            if password == 'skrrt':
                        print('Access
granted')
            else:
                        print('Try
again')
```

When writing code with multiple blocks, separate each one with a newline for readability.

Control Flow Statements

if Statement

If condition is true, execute the code in clause.

- the **if** keyword
- condition
- colon
- indented code block

The if statement is the first check in most control flow diagrams and is skipped if the condition is found to be false.

else Statement

If condition is true, execute the code. Else, execute that code.

- the **else** keyword
- colon
- indented code block

The else statement is the last resort in a control flow diagram and is only executed when the if statement is false.

elif Statement

If condition is true, execute the code. If else, execute this code.

- the **elif** keyword
- condition
- colon
- indented code block

The else-if statement is used when there are multiple clauses or checks to execute if the previous were found to be false.

All of these statements can be combined into large and complex control flow.

Control Flow Examples

if Statement

```
num = 42
if num == 42:
            print(‘Forty-
two’)



val = (2 > 3)
if val == True:
            print(‘contradict
ion’)
```

else Statement

```
num = 40
if num == 42:
            print(‘Forty-two’)
else:
            print(‘Some other num’)


max = 0
if a > b:
            max = a
else:
            max = b
```

elif Statement

```
num = 40

if num <= 25:
            print(‘Apples’)
elif num <= 50:
            print(‘Bananas’)
elif num <= 75:
            print(‘Cherries’)
else:
      print(‘Durians’)
```

while-Loop

Components

Execute the code in the clause so long as the while condition is true.

- the **while** keyword
- condition
- colon
- indented code block

The while loop behaves like an if statement except it jumps back to the start of the clause.

break Statement

Used to break out of a loop early (termination).

```
while True
            print('Type your
name')
            name = input()
            if name == 'your
name':
                        break
```

continue Statement

Used to restart the loop upon execution (immediate reiteration).

```
while True
            print('Type your
name')
            name = input()
            if name != 'your
name':
            continue
            else:
                        break
```

- While-loops are used to execute a block of code repeatedly.
- Three steps during execution
  - Initialization
  - Condition
  - Update

Control loops are the fundamental basics of designing algorithms.

for-Loop

Components

Execute the code in the clause for the specified number of times

- the **for** keyword
- variable name
- the in keyword
- call to range function
- colon
- indented code block

The range function

Execute the code in the clause for the specified number of times.

The range function can passed three arguments.

- 1 arg (n) from 0 to n
- 2 arg (a,b) from a to b
- 3 arg (a,b,c) from a to b in intervals of c

- For-loops are used for a limited set of iterations.
- Three steps during execution
  - Initialization
  - Condition
  - Update

Nested loops can be made by enclosing one or multiple loops in another loop.

for-Loop

- Break and continue statements can also be used in for-loops

Examples

```
# will print out ‘Hello’ five times
for i in range(5):
            print(‘Hello’)


# will add up all numbers between 0 and 100
total = 0
For num in range(101):
            total = total + num
print(total)
```

Equivalent while-loop

A for-loop can be written in terms of a while-loop (less concise, however).

```
print(‘My name is’)
i = 0
while i < 5:
            print(‘Hello (‘ + str(i) +
’)’)
            i = i + 1
```

Functions

- Python offers built-in functions
  - input()
  - print()
  - len()
- But you can write your own!
- Functions group code that gets executed many times
- Avoid code duplications

```
# name this: helloFunc1.py

def hello():
        print('Howdy!')
        print('Hello
there!')
        print('Salutation
s!')

hello()
hello()
```

- First line is def statement
- **hello()** lines are function calls
  - Program will jump to the function definition and execute code

Like variables, remember to use clear, understandable names for your functions.

def Statements with Parameters

- Pass in values (or arguments)
- In math, a function **f(x) = sin(x)** can be passed an arg of **π**.
  - **f(π) = sin(π) = 0**
- Values stored in parameters are destroyed after function calls!

```
# name this: helloFunc2.py

def hello(name):
          print('Hello ' +
name)

print('What is your name?')
myName = input()
hello(myName)
```

Always trace through your code by hand if you are struggling with debugging.

Return values

- We pass a str arg ‘Hello’ to **len()**.
- The fn call evaluates to an int value 5
- The fn returns the value back to us.

return Statement

Value that a function ultimately evaluates to.

- the **return** keyword
- value or expression that the function should return

```
# name this magic8ball.py

import random

def getAnswer(answerNum):
                if answerNum == 1:
                                return ‘It is certain’
                elif answerNum == 2:
                                return ‘It is
decidedly so’
                elif answerNum == 3:
                                return ‘Yes’
                elif answerNum == 4:
                                return ‘Without a
doubt’
                elif answerNum == 5:
                                return ‘Ask again
later’
                elif answerNum == 6:
                                return ‘Concentrate
and ask again’
                elif answerNum == 7:
                                return ‘My reply is
no’
                elif answerNum == 8:
                                return ‘Outlook not so
good’
```

When writing a function, think about what the final answer (return value) should be.

Local & Global Scope

- Parameters and variables assigned in a called fn exist in its **local scope**
- Variables that are assigned outside **ALL** fn's exist in **global scope.**
- Scopes are containers that destroy its belongings when it terminates.
- The **global** statement can manually force global scope of a variable.

Rules for Scope

- Code in the global scope cannot use any local variables.
- Local scope can access global variables.
- Code in a fn's local scope cannot use variables in any other local scope.
- You can use the same name for different variables so long as they are in different scopes.

Local and global scopes help narrow down where pesky bugs might reside.

Local & Global Scope

case 0

```
def spam():
            eggs =
123

spam()
print(eggs)
```

ERROR: eggs only exists in the local scope of **spam()** is called.

case 1

```
def spam():
            eggs =
123
            bacon()
            print(eg
gs)
def bacon():
            ham = 36
            eggs = 0

spam()
```

NO ERROR.

case 2

```
def spam():
            print(eg
gs)

eggs = 42
spam()
```

NO ERROR: Python considers print's arg as a reference to the global variable **eggs.**

case 3

```
def spam():
            eggs = 'local
eggs'
            print(eggs)
def bacon():
            eggs = 'local
bacon'
            print(eggs)
            spam()
            print(eggs)

eggs = 'global'
bacon()
print(eggs)
```

Final Program

- Write a code block for a “Guess the number” game.

```
I am thinking of a number between 1 and 20.
Take a guess.
10
Your guess is too low.
Take a guess.
15
Your guess is too low.
Take a guess.
17
Your guess is too high.
Take a guess.
16
Good job! You guessed it in 4 guesses!.
```

```
# this is a guess the number game
import random
secretNum = random.randint(1, 20)
guessesTaken = 0
print('I am thinking of a number between 1 and 20.')

# ask the player to guess a number
# player's current number of guesses is tracked
# HINT: Use an infinite loop! (while True:)

#########################################################
#                      YOUR CODE GOES HERE
#
#                          HAVE FUN ^_^
#
#########################################################

if guess == secretNum:
            print('Good job! You guessed it in '
            + str(guessesTaken) + ' guesses!')
```

Thank you!

Contact Info

Email: vinhkhang314@gmail.com