History of Python

- Invented in the Netherlands, early 90s by Guido van Rossum
- Named after Monty Python
- Open sourced from the beginning
- Managed by Python Software Foundation
- Considered a scripting language, but is much more
- Scalable, object oriented and functional from the beginning
- Used by Google from the beginning

Python's Benevolent Dictator For Life

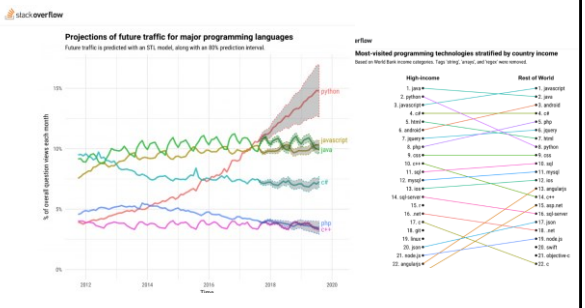
"Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas.

My office ... would be closed, but I had a home computer, and not much else on my hands.

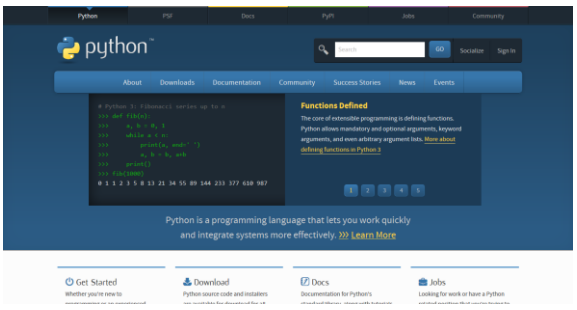
I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers.

I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of *Monty Python's Flying Circus*)." - Guido van Rossum

Python's place in the market



https://www.python.org



RUNNING PYTHON

Python interpreter

Example of Python interpreter running in interactive mode

Python interpreter

- Typical Python implementations offer both an interpreter and compiler
- Example of printing “Hello World” using Python interpreter:

```
>>> print ('Hello World')
Hello World
>>>
```


Installing

- Python is pre-installed on most Unix systems, including Linux and MAC OSX
- The pre-installed version may not be the most recent
- Two “latest versions”:
  - 2.7 (final version, no new major releases)
  - 3.x (present and future of the language)
- Python 3 is not backwards compatible
- Download from http://python.org/download/

Python IDEs

- An Integrated Development Environment (IDE) is an application that facilitates application development
- An IDE typically consists of a **source code editor**, **build automation tools** and a **debugger**
- There are many IDEs, such as:
  - IDLE
  - PyCharm
  - Wing IDE
  - …

9

IDLE Development Environment

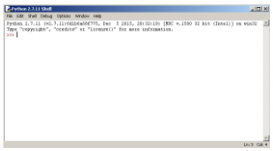
- IDLE is the "official" IDE distributed with Python
- Written in Python with the Tkinter GUI package
- Multi-window text editor with syntax highlighting, auto-completion, smart indent and other features
- Python shell with syntax highlighting, line recall, and more…
- Integrates a debugger with stepping, persistent breakpoints…

10

Running Python Interactively

On the interpreter…

```
>>> 3+3
6
```



- Python prompts with '>>>'.
- To exit Python
  - Type in prompt: exit()

11

PROGRAM DESGIN

12

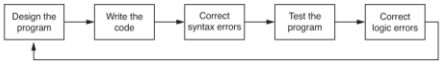
Designing a Program

- Programs must be designed before they are written
- Program development cycle:
  - Design the program
    - Plan system before programming
  - Write the code
    - With design created, begin coding
  - Correct syntax errors
    - When syntax errors corrected, program can be compiled
  - Test the program
    - Check program for *logic errors* . These errors will not prevent compiling of program, but may yield incorrect result
  - Correct logic errors
    - Programmers debug code. This refers to the correction of logic errors

13

Designing a Program (cont'd.)

- Design is the most important part of the program development cycle



- Understand the task that the program is to perform
  - Get an understanding of what the program is supposed to do
  - Ask questions about program details
  - Create one or more software requirements

14

Designing a Program (cont'd.)

- Determine the steps that must be taken to perform the task
  - Break down required task into a series of steps
  - Create an algorithm, listing logical steps that must be taken
- Algorithm: set of well-defined logical steps that must be taken to perform a task

15

Algorithm example

- Suppose you have been asked to write a program to calculate and display the gross pay for an hourly paid employee.
- What steps would you take ?

16

Algorithm example

- Suppose you have been asked to write a program to calculate and display the gross pay for an hourly paid employee.
- Here are the steps that you could take:
  1. Get the number of hours worked
  2. Get the hourly pay rate
  3. Multiply the number of hours worked by the hourly pay rate
  4. Display the result of the calculation that was performed in step 3

17

Pseudocode

- Pseudocode: fake code
  - Informal language that has no syntax rule
  - Not meant to be compiled or executed
  - Used to create model program
    - No need to worry about syntax errors, can focus on program's design
    - Can be translated directly into actual code in any programming language
- Example:
  - Input the hours worked
  - Input the hourly pay rate
  - Calculate gross pay as hours worked multiplied by pay rate
  - Display the gross pay

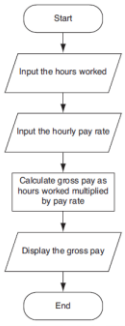
18

Flowcharts

- Flowchart: diagram that graphically depicts the steps in a program
  - Ovals are terminal symbols
  - Parallelograms are input and output symbols
  - Rectangles are processing symbols
  - Symbols are connected by arrows that represent the flow of the program

19

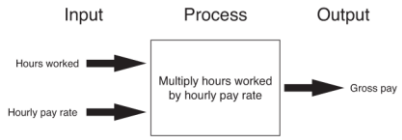
Flowchart for pay calculation



20

Input, Processing, and Output

- Typically, computer performs three-step process
  - Receive input
    - Input: any data that the program receives while it is running
  - Perform some process on the input
    - Example: mathematical calculation
  - Produce output



21

Displaying Output with the `print` Function

- Function…
  - … is a piece of pre-written code that performs an operation
  - Python has many built-in functions that perform various operations
- Example:
- **`print`** function
  - displays output on the screen

```
>>> print('Hello world')
    Hello world
>>>
```

22

Displaying Output with the `print` Function

- Argument
  - data given to a function
  - Example: data that is printed to screen

```
>>> print('Hello world')
    Hello world
>>>
```

- Statements
  - … in a program execute in the order that they appear
  - From top to bottom

```
print('Joe Smith')
print('123 Crest St.')
print('Paramus, NJ 07652')
```

23

Strings and String Literals

- String
  - sequence of characters that is used as data
- String literal
  - string that appears in actual code of a program
  - Must be enclosed in single (') or double (") quote marks
  - String literal can be enclosed in triple quotes (''' or """)
    - Enclosed string can contain both single and double quotes and can have multiple lines

```
print('Joe Smith')
print('123 Crest St.')
print('Paramus, NJ 07652')
```

24

Comments

- Comments
  - notes of explanation within a program
  - Ignored by Python interpreter
    - Intended for a person reading the program's code
  - Begin with a # character

```
# This program displays a person's
# name and address
print('Joe Smith')
print('123 Crest St.')
print('Paramus, NJ 07652')
```

- End-line comment
  - appears at the end of a line of code
  - Typically explains the purpose of that line

Comments

- End-line comment
  - appears at the end of a line of code
  - Typically explains the purpose of that line
- Example:

```
print('Joe Smith')                # Display name
print('123 Crest St.')            # Display address
print('Paramus, NJ 07652')        # Display city, state, zipcode
```


Variables

- Variable
  - name that represents a value stored in the computer memory
  - Used to access and manipulate data stored in memory
  - A variable references the value it represents
- Assignment statement
  - used to create a variable and make it reference data
  - General format is `variable = expression`
    - Example: `age = 29`
    - Assignment operator: the equal sign (=)

Variables (cont'd.)

- In assignment statement, variable receiving value must be on left side
- A variable can be passed as an argument to a function
  - Variable name should not be enclosed in quote marks
- You can only use a variable if a value is assigned to it

Keywords (reserved words)

- Keywords are words that are reserved for the Python programming language
- You should avoid using these words as variables

and	del	from	not	while
as	elif	global	or	with
assert	else	if	pass	yield
break	except	import	print	
class	exec	in	raise	
continue	finally	is	return	
def	for	lambda	try	

29

Variable Naming Rules

- Rules for naming variables in Python:
  - Variable name cannot be a Python key word
  - Variable name cannot contain spaces
  - First character must be a letter or an underscore
  - After first character may use letters, digits, or underscores
  - Variable names are case sensitive
- Variable name should reflect its use

30

Displaying Multiple Items with the `print` Function

- Python allows one to display multiple items with a single call to `print`
  - Items are separated by commas when passed as arguments
  - Arguments displayed in the order they are passed to the function
  - Items are automatically separated by a space when displayed on screen

31

Numeric Data Types, Literals, and the `str` Data Type

- Data types: categorize value in memory
  - e.g., int for integer, float for real number, str used for storing strings in memory
- Numeric literal: number written in a program
  - No decimal point considered int, otherwise, considered float
- Some operations behave differently depending on data type

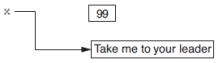
33

Reassigning a Variable to a Different Type

- A variable in Python can refer to items of any type

Figure 2-7 The variable x references an integer

Figure 2-8 The variable x references a string



34

Reading Input from the Keyboard

- Most programs need to read input from the user
- Built-in `input` function reads input from keyboard
  - Returns the data as a string
  - Format: *`variable = input(prompt)`*
    - `prompt` is typically a string instructing user to enter a value
  - Does not automatically display a space after the prompt

35

Reading Numbers with the `input` Function

- `input` function always returns a string
- Built-in functions convert between data types
  - `int(item)` converts *`item`* to an `int`
  - `float(item)` converts *`item`* to a `float`
  - Nested function call: general format:
    `function1(function2(argument))`
    - value returned by function2 is passed to function1
  - Type conversion only works if item is valid numeric value, otherwise, throws exception

36

Performing Calculations

- Math expression: performs calculation and gives a value
  - Math operator: tool for performing calculation
  - Operands: values surrounding operator
    - Variables can be used as operands
  - Resulting value typically assigned to variable
- Two types of division:
  - `/` operator performs floating point division
  - `//` operator performs integer division
    - Positive results truncated, negative rounded away from zero

```
>>> 5/2
2.5
>>> 5//2
2
>>>
```

37

Operator Precedence and Grouping with Parentheses

- Python operator precedence:
  1. Operations enclosed in parentheses
     - Forces operations to be performed before others
  2. Exponentiation (**)
  3. Multiplication (*), division (/ and //), and remainder (%)
  4. Addition (+) and subtraction (-)
- Higher precedence performed first
  - Same precedence operators execute from left to right

38

The Exponent Operator and the Remainder Operator

- Exponent operator (`**`): Raises a number to a power
  - `x ** y` = x^y
- Remainder operator (`%`): Performs division and returns the remainder
  - a.k.a. modulus operator
  - e.g., `4%2=0`, `5%2=1`
  - Typically used to convert times and distances, and to detect odd or even numbers

39

Converting Math Formulas to Programming Statements

- Operator required for any mathematical operation
- When converting mathematical expression to programming statement:
  - May need to add multiplication operators
  - May need to insert parentheses

40

Mixed-Type Expressions and Data Type Conversion

- Data type resulting from math operation depends on data types of operands
  - Two `int` values: result is an `int`
  - Two `float` values: result is a `float`
  - `int` and `float`: `int` temporarily converted to `float`, result of the operation is a `float`
    - Mixed-type expression
  - Type conversion of `float` to `int` causes truncation of fractional part

41

Breaking Long Statements into Multiple Lines

- Long statements cannot be viewed on screen without scrolling and cannot be printed without cutting off
- Multiline continuation character (\): Allows to break a statement into multiple lines
  - Example:

```
print('my first name is',\
 first_name)
```

42

More About Data Output (cont'd.)

- Special characters appearing in string literal
  - Preceded by backslash (\)
    - Examples: , horizontal tab (\t)
  - Treated as commands embedded in string
- When + operator used on two strings in performs string concatenation
  - Useful for breaking up a long string literal

44

Formatting Numbers

- Can format display of numbers on screen using built-in `format` function
  - Two arguments:
    - Numeric value to be formatted
    - Format specifier
  - Returns string containing formatted number
  - Format specifier typically includes precision and data type
    - Can be used to indicate scientific notation, comma separators, and the minimum field width used to display the value

45