

Introduction to Python

Robert Smith

7/11/2023

What is Python?

- Named after Monty Python's Flying Circus
- High-Level Language
- General Purpose Language
- Multi Paradigm
- Dynamic Typing
- Strongly Typed
- Garbage Collector

Design Philosophy

- Simpler, less cluttered syntax and grammar
- Readable
- Programmer has choice in coding methodology
- Everything has a necessary purpose
- “In Python, every symbol you type is essential.” – Guido Van Rossum

The Zen of Python

- 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.
- 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!

Python Syntax, Semantics, and Basics

Indentation

- Python uses whitespace to denote blocks
- Indenting a logical line will place it in the body of a “parent’s” block
- The “parent” line of the block will have a colon (:) at the end
- The next de-indented line will “end” the block
- The visual structure accurately represents the semantic structure

```
1 x = int(input("Enter number of minutes: "))
2 if x < 5:
3     print("NO")
4 else:
5     print("YES")
6     print("Potato is cooked")
7
```

```
1 x = int(input("Enter number of minutes: "))
2 if x < 5:
3     print("NO")
4 else:
5     print("YES")
6 print("Potato is cooked")
7
```

Comments

- A comment starts with the hash (#) character
- It marks the end of a logical line (no code will be executed after it on a line)
- Ignored by the syntax
- One line only per #

```
1  # This is a comment
2  age = 23 # comment after code
3  # another comment
4  # comment with code but no execution age = 40
5
```

Docstrings

- “Similar” to comments and often seen as “multi-line” comments
- A docstring is a string literal that occurs as the first statement in a module, function, class, or method definition. Such a docstring becomes the `__doc__` special attribute of that object.
- See [PEP 257 – Docstring Conventions](#)

```
# Note the this is not how you use them
"""One line docstring"""

"""
Multi Line
Docstring
"""
```

```
def complex(real=0.0, imag=0.0):
    """Form a complex number.

    Keyword arguments:
    real -- the real part (default 0.0)
    imag -- the imaginary part (default 0.0)
    """
    if imag == 0.0 and real == 0.0:
        return complex_zero
    ...
```


Keywords

- There are 35 “hard” [keywords](#)

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

- There are 3 “soft” keywords
 - match, case, and _

Literals

- Literals are notations for constant values of some built-in types
- Broader Categories
 - String Literals
 - Bytes Literals
 - Numeric Literals

String Literals

- Can be enclosed in either single or double quotes ‘ or “

```
name = "Robert"  
l_name = 'Smith'
```

- Escape sequences are interpreted according to rules used by Standard C
- May be optionally prefixed with ‘r’ or ‘R’ for raw strings (no escapes)

Escape Sequence	Meaning	Notes
\<newline>	Backslash and newline ignored	(1)
\\	Backslash (\)	
\'	Single quote (')	
\"	Double quote (")	
\a	ASCII Bell (BEL)	
\b	ASCII Backspace (BS)	
\f	ASCII Formfeed (FF)	
\n	ASCII Linefeed (LF)	
\r	ASCII Carriage Return (CR)	
\t	ASCII Horizontal Tab (TAB)	
\v	ASCII Vertical Tab (VT)	
\ooo	Character with octal value <i>ooo</i>	(2,4)
\xhh	Character with hex value <i>hh</i>	(3,4)

Formatted String Literals

- String literals prefixed with 'f' or 'F'
- Can contain replacement fields which are expressions delimited by curly braces {}
- an equal sign '=' may be added after the expression

```
a, b = 10, 3.14
name = "Robert"
l_name = 'Smith'

print(f"{name=}")
print(f"b is {b:.2f}")
```

```
• (base) jovyan@jupyter-rwsmith:~/SummerPython$ python test.py
name='Robert'
b is 3.14
```

Bytes Literals

- Bytes literals are always prefixed with 'b' or 'B'
- Produce an instance of bytes type instead of str type
- May only contain ASCII characters
- bytes with a numeric value of 128 or greater must be expressed with escapes.
- May be optionally prefixed with 'r' or 'R' for raw strings (no escapes)

```
b'Hello world'  
b'\x7f\x45\x4c\x46\x01\x01\x01\x00'
```

Numeric Literals

- Three types
 - Integers
 - Floating point numbers
 - Imaginary numbers

Integer Literals

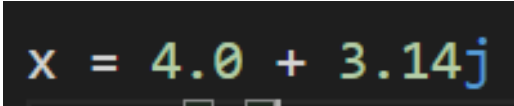
- Whole numbers
- No limit for length apart from what can be stored in available memory
- Underscores are ignored in them
- Leading 0's not allowed in non-zero decimal number

Numeric Literals cont.

Floating Point Literals

- Integer and exponent parts are always interpreted using base 10
- Allowed range of floating point literals is implementation dependent
- Underscores are supported for digit grouping.

Imaginary Literals

- Uses 'j' or "J" 
- Yields complex number with a real part of 0.0
- Represented as a pair of floating point numbers
- Same range restrictions

Mathematical Operators

- ******
 - Power Operator
 - Same semantics as pow()
- **+X, -X**
 - unary positive/negative
- ***, /, //, %**
 - multiplication, division, floor division, remainder
- **+, -**
 - Addition and Subtraction

Precedence from High -> Low

- ******
- **+X, -X**
- ***, /, //, %**
- **+, -**
- Python evaluates expressions from left to right
- while evaluating an assignment, the right-hand side is evaluated before the left-hand side

Variables and Assignment Statements

- A variable is a reference to an object
- Python *values* NOT variables contain/carry the type information
- A variable name is a generic reference holder
- A variable can be rebound to another object of any type
- Variables are assigned a value using an assignment statement
- Generic form: *var = expression*

```
1 name = "Bob"
2 age = 23
3 gpa = 3.14
4
5 name = input("Enter your name: ")
6 value = age * 3
7 final_gpa = gpa + value / 100 * 1.23
```

- Python supports multiple item assignment

```
a, b = 10, 3
```

Functions

- A function definition defines a user-defined function object
- Definition DOES NOT execute the body
- Execution occurs when function is called
- Can be wrapped by decorator expressions (more on this at a later date)

- General Form

```
# Function header
def myfunc(parameter, list, ...):
    # Function body
    # that does stuff
```

Functions cont.

- A function may have 0 or more parameters
- Parameters may have default values
 - If a parameter has a default value, then all following parameters must also have a default value
- The return statement allows you to return end a function and optionally return one or more values
- Multiple return statements are allowed

Input

- The *input()* function is built-in
- Has optional *prompt* parameter
 - *input(prompt)*
- Reads line from input
- Converts line to a string
- Returns converted line

```
name = input("Enter your name: ") # returns a string
age = input("Enter your age: ") # returns A STRING STILL
```

Output

- *print()* function
- *print(*objects, sep=' ', end='\n', file=None, flush=False)*
- *sep* = " " is the separator (defines what , separator becomes when multiple objects are printed)
- *end* = '\n' is what prints at the end of the objects (default is a newline)
- *file*=None defaults to sys.stdout

Python Data Types and Data Model

- bool
- int
- float
- complex
- bytes
- str
- function
- class