

# The Absolute Basics

## Chapter 4

# This chapter covers

- Indenting and block structuring
- Differentiating comments
- Assigning variables
- Evaluating expressions
- Using common data types
- Getting user input
- Using correct Pythonic style

```
1  # This is Python code. (Yea!)
2  n = 9
3  r = 1
4  while n > 0:
5      r = r * n
6      n = n - 1
```

## Indentation and block structuring

- Python differs from most other programming languages because it uses whitespace and indentation to determine block structure.

# Advantages of indentation

- It's impossible to have missing or extra braces. You never need to hunt through your code for the brace near the bottom that matches the one a few lines from the top.
- The visual structure of the code reflects its real structure, which makes it easy to grasp the skeleton of code just by looking at it.
- Python coding styles are mostly uniform. In other words, you're unlikely to go crazy from dealing with someone's idea of aesthetically pleasing code. Everyone's code will look pretty much like yours.

```
9    # Assign 5 to x
10   x = 5
11   x = 3 # Now x is 3
12   x = "# This is not a comment"
```

## Differentiating comments

- For the most part, anything following a # symbol in a Python file is a comment and is disregarded by the language.
- The obvious exception is a # in a string, which is just a character of that string.

# Variables and assignments

- In Python, unlike in many other computer languages, neither a variable type declaration nor an end-of-line delimiter is necessary.
- The line is ended by the end of the line.
- Variables are created automatically when they're first assigned.
- Python variables can be set to any object, whereas in C and many other languages, variables can store only the type of value they're declared as.
- The following is perfectly legal Python code

```
14 x = "Hello"  
15 print(x)  
16  
17 x = 5  
18 print(x)
```

# Expressions

- Arithmetic and similar expressions.
- Standard rules of arithmetic precedence apply. If you'd left out the parentheses in the last line, the code would've been calculated as  $x + (y / 2)$ .

1     $x = 3$

2     $y = 5$

3     $z = (x + y) / 2$



# TRY THIS

---

- In the Python shell, create some variables.
- What happens when you try to put spaces, dashes, or other nonalphanumeric characters in the variable name?
- Play around with a few complex expressions, such as  $x = 2 + 4 * 5 - 6 / 3$ . Use parentheses to group the numbers in different ways and see how the result changes compared with the original ungrouped expression.



```
39 x = "Hello, World"
40
41 x = "\tThis string starts with a \"tab\"."
42 x = "This string contains a single backslash(\\)."
43
44 x = "Hello, World"
45 x = 'Hello, World'
46
47 x = "Don't need a backslash"
48 x = 'Can\'t get by without a backslash'
49 x = "Backslash your \" character!"
```

# Strings

Python, like most other programming languages, indicates strings using double quotes.

```
4  # print age to output
5  print(age)
6
7  # numbers
8  print(5 + 2 - 3 * 2)
9  print(5 / 2)      # floating 2.5
10 print(5 / 2.0)    # also floating 2.5
11 print(5 // 2)     # integer result 2
12 print(3000000000) # a large number
```

# Numbers

Python offers four kinds of numbers: integers, floats, complex numbers, and Booleans.



# Built-in numeric functions

- Python provides the following number-related functions as part of its core:
  - abs
  - divmod
  - float
  - hex
  - int
  - max
  - min
  - oct
  - pow
  - round

# Advanced numeric functions

---

acos	asin	atan	ceil	cos
cosh	e	exp	fabs	floor
fmod	frexp	hypot	ldexp	log
log10	mod	pi	pow	sin
sinh	sqrt	tan	tanh	

---

- from math import \*



# TRY THIS

---

- In the Python shell, create some string and number variables (integers, floats, and complex numbers).
- Experiment a bit with what happens when you do operations with them, including across types.
- Can you multiply a string by an integer, for example, or can you multiply it by a float or complex number?
- Also load the math module and try a few of the functions; then load the cmath module and do the same. What happens if you try to use one of those functions on an integer or float after loading the cmath module? How might you get the math module functions back?

# The None value

- Python has a special basic data type that defines a single special data object called None.
- As the name suggests, None is used to represent an empty value.
- None is often useful in day-to-day Python programming as a placeholder to indicate a point in a data structure where meaningful data will eventually be found, even though that data hasn't yet been calculated.

```
1  # getting input from user
2  name = input("Name? ")
3  print(name)
4
5  age = int(input("Age? "))
6  print(age)
```

## Getting input from the user

- Use the input() function to get input from the user.



# TRY THIS

---

- Experiment with the `input()` function to get string and integer input. Using code similar to the previous code, what is the effect of not using `int()` around the call to `input()` for integer input?
- Can you modify that code to accept a float - say, 28.5?
- What happens if you deliberately enter the wrong type of value? Examples include a float in which an integer is expected and a string in which a number is expected - and vice versa.



# Built-in operators

- Python provides various built-in operators, from the standard (+, \*, and so on) to the more esoteric, such as operators for performing bit shifting, bitwise logical functions, and so forth.
- Most of these operators are no more unique to Python than to any other language.

# Basic Python style

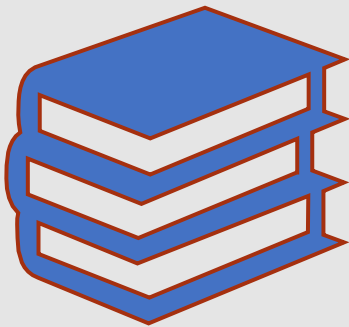
Situation	Suggestion	Example
Module/package names	Short, all lowercase, underscores only if needed	imp, sys
Function names	All lowercase, underscores_for_readability	foo(), my_func()
Variable names	All lowercase, underscores_for_readability	my_var
Class names	CapitalizeEachWord	myclass
Indentation Comparison	Four spaces per level, no tabs  Don't compare explicitly to True or False	if my_var: if not my_var:

# Quick Check

- Which of the following variable and function names do you think are not good Pythonic style? Why?

bar()	varName	VERYLONGVARNAME	foobar
longvarname	foo_bar()	really_very_long_var_name	

# Summary



- The basic syntax summarized above is enough to start writing Python code.
- Python syntax is predictable and consistent.
- Because the syntax offers few surprises, many programmers can get started writing code surprisingly quickly.