Introduction to Python Part II

Beste Filiz Yuksel

Debugging

As you have probably already experienced, errors may often occur when programming.

The process of tracking down and fixing errors, or bugs, in code is called **debugging**.

Code and Test in Small Pieces

Don’t write a 25-line program, then test the whole thing.

Write 2-3 lines, test it.

Write 3 more lines, test all 6

Write 3 more lines, test all 9

Write 3 more lines, test all 12

Write 3 more lines, test all 15

…

Code and Test in Small Pieces

If you wait until all code is written to test it:

- Programs will fail in obscure, hard-to-diagnose ways.
- You will spend much time looking for bugs
- Some bugs will stay hidden for a long time
- You will blame Python for the error

Different Kinds of Coding Errors

Syntax errors

```
name = input(“Enter your name)     # Error! Missing closing quote
```

Runtime errors

```
a = 0
x = 42 / a
print(x)                 # Error! ZeroDivisionError
```

Logic or Semantic errors

```
miles = 42
km = miles * 1.4         # Wrong conversion (no error msg)
```

Kinds of Errors in Python

Parse error: incorrect code syntax

Type error: incompatible values

Name error: variable not yet defined

Value error: gave function invalid input

See Chapter 3 of How To Think Like A Computer Scientist, particularly section 3.4:

http://interactivepython.org/runestone/static/thinkcspy/Debugging/KnowyourerrorMessages.html

Syntax Errors

Each programming language (like each natural language e.g. commas, apostrophes, capitalization etc) has its own rules, keywords, and operators - i.e. *syntax.*

When learning a programming language, you must learn the syntax rules.

When using natural languages, humans often violate the syntax rules, and other people can still understand them.

With a programming language, even one violation of the syntax rule will be cause the interpreter or compiler to stop executing the program.

Syntax Errors

Once all syntax errors have been corrected the program can be compiled and translated into a machine language program or executed by an interpreter, depending on the language being used.

In order to overcome/avoid syntax errors, become familiar with the Python keywords (see Table 1.2 in Gaddis) and general syntax rules which we have started to cover.

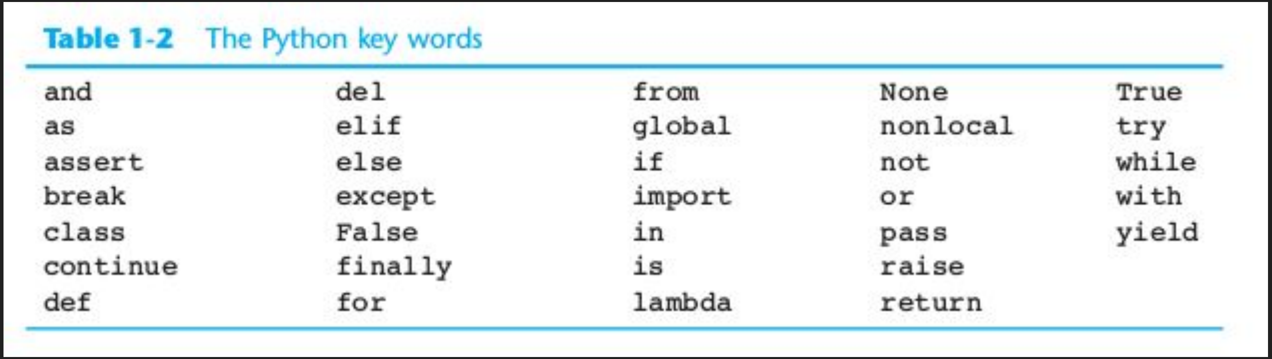


Table 1-2 The Python key words

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

Logic Errors

Once your program can run, it may still contain a logic error.

A logic error is a mistake that does not prevent the program from running but causes it to produce *an incorrect result.*

e.g. mathematical mistakes are common causes of logic errors.

In order to avoid/overcome logic errors, it is important to understand the task that the program is to perform and to think through the logic, determining the steps that must be taken.

Algorithm

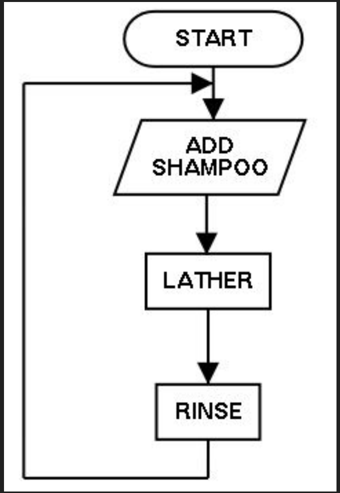
This will lead you to develop an ***algorithm***.

An algorithm is a set of well-defined logical steps that must be taken to perform a task.

Famous example of what *not* to do - Shampoo Algorithm:

➔ Lather
➔ Rinse
➔ Repeat

Endless loop



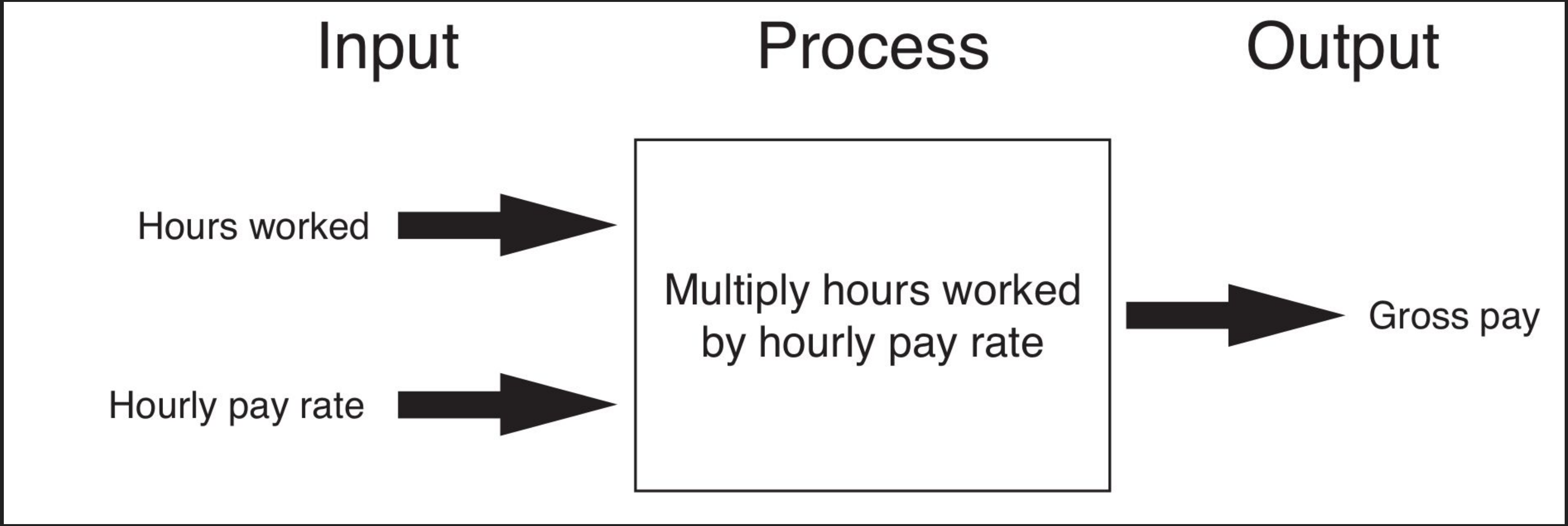
Algorithm

A programmer breaks down the task that a program must perform by creating an algorithm that lists all of the logical steps that must be taken.

e.g. Write a program to calculate and display the gross pay for an hourly paid employee.

- ➔ Get number of hours worked.
- ➔ Get hourly pay rate.
- ➔ Multiply number of hours by hourly pay rate.
- ➔ Display result.

Algorithm in terms of input, process, output



Remember output is using the `print( )` function!

Math Operators in Python

- Add: +
- Subtract: -
- Multiply: *
- Divide: / Integer divide: //
- Remainder: %
- Power (exponent): **

Integer Division   //

- ***Regular*** division results in decimal fractions (does true division)
  - >>> 12 / 4
  - 3.0   (not just 3)
  - >>> 12 / 5
  - 2.4
- ***Integer*** division // truncates any fractional part of result and only gives integer part
  - >>>12 // 5
  - 2

Remainder (modulo operation)

Remainder operator gives the remainder (what’s left) after division is performed

```
>>>16 % 5
1
>>>10 % 4
2
```

We’ll be using these for integers. If you are interested in modulo and floating point numbers in Python, take a look at https://stackoverflow.com/questions/14763722/python-modulo-on-floats

Operator Precedence in Python

1. Expressions in parentheses ( )
   - Inner-most first
2. Exponent/power **
3. Multiply, divide, remainder * / // %
4. Addition and subtraction + -

If precedence is equal: go left to right

Mixed-Type Math Expressions

- Data-type of result depends on data-types of operands
- Two `int` values: result is an `int` (except with division)
- Two `float` values: result is a `float`
- `int` and `float`: int temporarily converted to float, result is a `float`
- Converting `float` to `int` **truncates** fractional part

Comments in Python Programs

Explanatory notes embedded in program

Ignored by Python interpreter & compiler

Intended for person reading program code

Start with #

- Ends at end of line

```
# Ask user for bank balance
balance = int(input("Enter balance: "))


v = int(input("Enter velocity: ")# get velocity from user
```

Controlling Output in Python

- Quotes in output: single vs. double quotes
  - *print(‘The phrase “Cura Personalis” is Latin for “Care of the whole person” ’)*
- Print multiple items with one **print,** two ways:
  - *print(firstName, lastName)*
  - *print(firstName* + ‘, ’ + *lastName)* #try this also without the ‘,’
- Stop print from adding newline:
  - *print(“Hello”, end=“ ”)*
- Specify # of decimal places to print:
  - If *total* has been set to 24.56782102
  - print(“Total price: ”, format(total, ‘.2f’))
  - prints 24.57

Controlling Output in Python

- Escape character: print special characters:
    - \n advances output to next line, i.e. **newline** e.g. `print('hi \nbye')`
    - \t advances output to next tab position e.g. `print('Mon\tTues\tWed')`
    - \’ prints a single quote
    - \” prints a double quote
    - \\ prints a backslash
- Item separator
    - `print(1, 2, 3)                 => 1 2 3`
    - `print(1, 2, 3, sep=‘’)     => 123`
    - `print(1, 2, 3, sep=‘*’)   => 1*2*3`