Mastering Python

الدرس #1_2

Python’s Language Basics

اساسيات لغة بايثون

By:

Hussam Hourani

V1.0 - NOV 2018

Agenda

- Reserved words
- Python Comments
- Indentation
- Input/output
- Escape Sequence
- Data Types
- Data Types Examples & Casting
- Variables Names
- Assign Value to Multiple Variables
- Object References
- Arithmetic Operators
- Summary of Operations
- Getting Help
- Zen of Python

Reserved words

- The below is the reserved words in Python.

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

Python Comments

- Comments can be used to explain Python code.
- Comments can be used to make the code more readable.

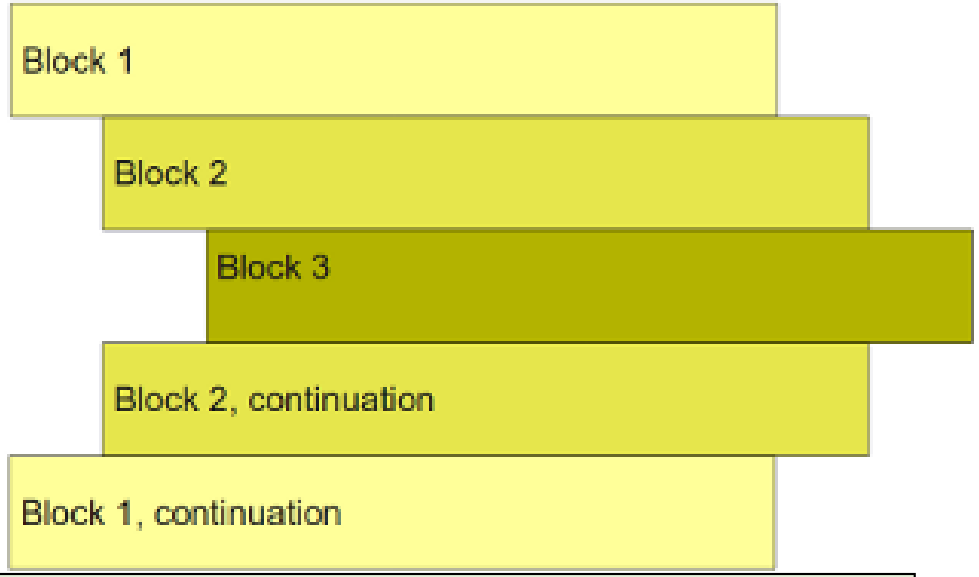
```
# this is the first comment
x = 1  # and this is the second comment
       # ... and now a third!


"""
If I really hate pressing `enter` and
typing all those hash marks, I could
just do this instead
"""
```

Indentation

Python programs get structured through indentation. Code blocks are defined by their indentation

```
import random
n = 20
to_be_guessed = int(n * random.random()) + 1
guess = 0
while guess != to_be_guessed:
    guess = int(input("New number: "))
    if guess > 0:
        if guess > to_be_guessed:
            print("Number too large")
        elif guess < to_be_guessed:
            print("Number too small")
    else:
        print("Sorry that you're giving up!")
        break
print("Congratulation. You made it!")
```




```
from math import sqrt
n = input("Maximum Number? ")
n = int(n)+1
for a in range(1,n):
    for b in range(a,n):
        c_square = a**2 + b**2
        c = int(sqrt(c_square))
        if ((c_square - c**2) == 0):
            print(a, b, c)
```

https://www.python-course.eu/python3_blocks.php

output

```
print ("Hello")
```

Output

```
Hello
```

```
print("hello")
print("hello")
```

Output

```
hello
hello
```

```
print('hello', 'world')
```

Output

```
hello world
```

```
print('hello', 'world', sep='')
```

Output

```
helloworld
```

```
print('hello', 'world', sep='\n')
```

Output

```
hello
world
```

```
print('hello\nworld')
```

Output

```
hello
world
```

```
print('hello', 'world', end=' ')
print('hello', 'world')
```

Output

```
hello world hello world
```

```
print('home', 'user', 'documents', sep='/')
```

Output

```
home/user/documents
```

output

```
print("Orange"*5)
```

Output

```
OrangeOrangeOrangeOrangeOrange
```

```
print("The Numbers are : %5d: %010d" % (5,-20))
```

Output

```
The Numbers are :      5: -000000020
```

```
print("Welcome %s" % 'Python')
print("Actual Number = %d" %15)
print("Float of the number = %f" %15)
print("Exponential equivalent of the number = %e" %15)
print("%f" % 5.1234567890)
print("%.3f" % 5.1234567890)
print("%015.5f" % 5.1234567890)
```

Output

```
Welcome Python
Actual Number = 15
Float of the number = 15.000000
Exponential equivalent of the number = 1.500000e+01
5.123457
5.123
000000005.12346
```

```
print ('{:3}  {:6}  {:10}  {:12}'.format('a',
'bc', 'def', 'gh'))
```

Output

```
 a   bc      def         gh
```

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""

print (word, sentence, paragraph)
```

Output

```
word This is a sentence. This is a paragraph.
It is
made up of multiple lines and sentences.
```

Escape Sequence

Escape Sequence	“Escaped” Interpretation	**Example**
\a	ASCII Bell (BEL)	
\b	ASCII Backspace (BS)	
\n	ASCII Linefeed (LF)	
\t	ASCII Horizontal Tab (TAB)	>>> print("a\tb") : a b
\'	Terminates string with single quote opening delimiter	>>>print('Orange\'s Academy') Orange's Academy

Input

```
age = input("What is your Age? ")
print(age)
print ( type(age)
```

Output

```
What is your Age? 15
15
<class 'str'>
```

```
age = int(input(" What is your Age? "))
print(age)
print ( type(age)
```

Output

```
 What is your Age? 15
15
<class 'int'>
```

```
cities_canada = eval(input("Largest cities in
Canada: "))
print(cities_canada)
print(type(cities_canada))
```

Output

```
Largest cities in Canada: ["Toronto",
"Montreal", "Calgara", "Ottawa"]
['Toronto', 'Montreal', 'Calgara', 'Ottawa']
<class 'list'>
```

Data Types

Type	Python Representation	Example
Text Type:	str	“Orange”, ‘Acadimy’
Numeric Types:	int, float, complex	182762554256782999 12.3455 2+4j
Sequence Types:	list, tuple, range	
Mapping Type:	dict	
Set Types:	set, frozenset	
Boolean Type:	bool	False, True
Binary Types:	bytes, bytearray, memoryview	

```
>>> type(1234)
<class 'int'>
>>> type(55.50)
<class 'float'>
>>> type(6+4j)
<class 'complex'>
>>> type("hello")
<class 'str'>
```

Data Types Examples & Casting

Example	Data Type
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex
`x = True`	Bool
`x = b"Hello"`	Bytes
X = range(6)	range

Casting Example	Result
int(12.1)	12
float(20)	20.0
`str(10)`	’10’
`int(‘100’)`	100
`bool(5)`	True
`bool(0)`	False
`bool(-1)`	True
bool(False), bool(None) bool("")bool(()), bool([]), bool({})	False

Variables Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

```
>>> age = 1
>>> Age = 2
>>> aGe = 3
>>> AGE = 4
>>> a_g_e = 5
>>> _age = 6
>>> age_ = 7
>>> _AGE_ = 8
```

```
>>> 76trombones = 'big parade'
SyntaxError: invalid syntax

>>> more@ = 1000000
SyntaxError: invalid syntax

>>> class = 'Advanced Theoretical Zymurgy'
SyntaxError: invalid syntax
```


https://www.w3schools.com/python/python_variables.asp

Assign Value to Multiple Variables

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

Output

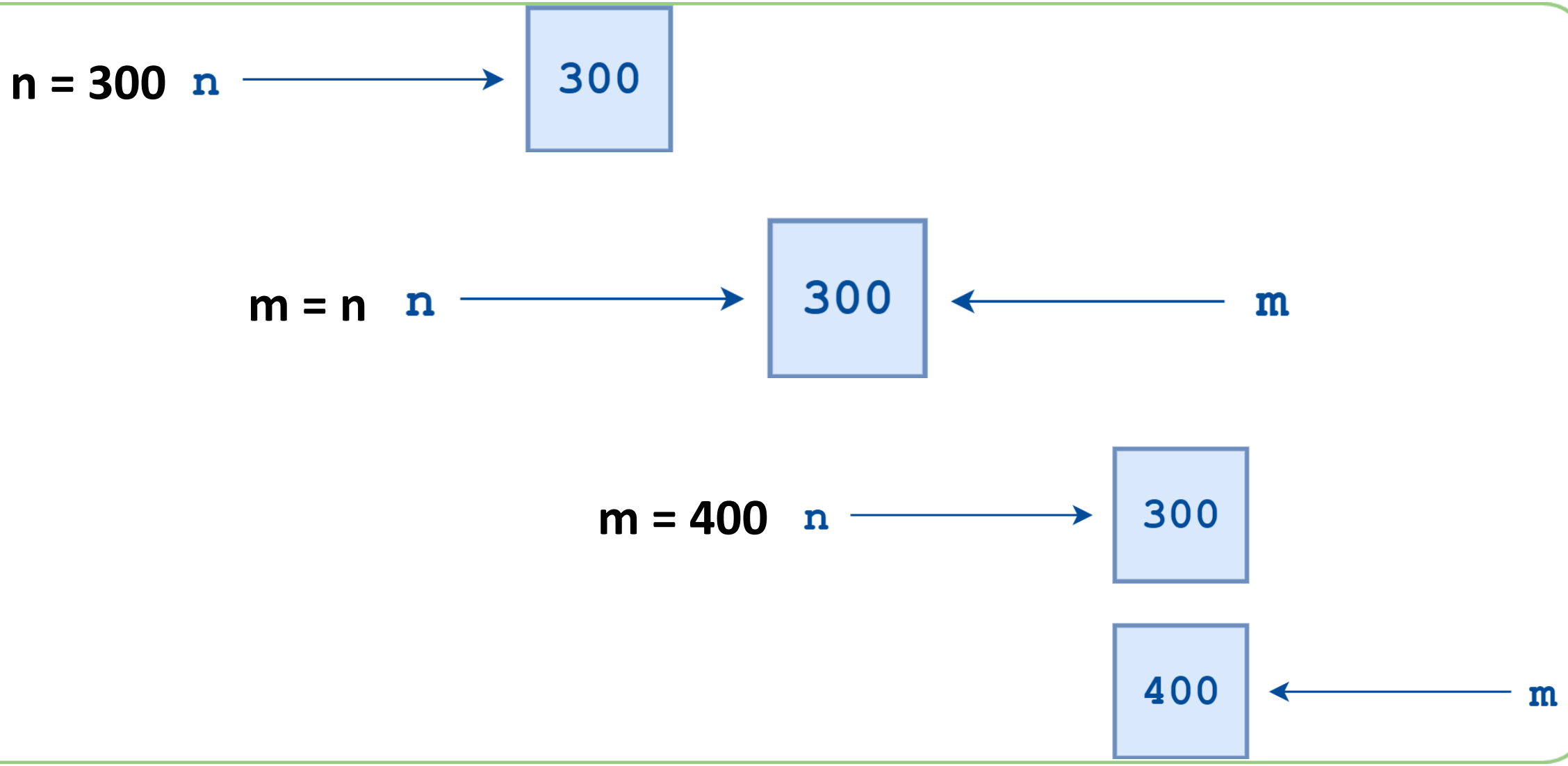
```
Orange
Banana
Cherry
```

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

Output

```
Orange
Orange
Orange
```

Object References



Arithmetic Operators

```
number = 1 + 2 * 3 / 4.0
print(number)
```

Output

```
2.5
```

```
n = 11 / 3
print(n)
```

Output

```
3.6666666666666665
```

```
n = 11 // 3
print(n)
```

Output

```
3
```

```
remainder = 11 % 3
print(remainder)
```

Output

```
2
```

```
helloworld = "hello" + " " + "world"
print(helloworld)
```

Output

```
hello world
```

Summary of Operations

Boolean Operations

Operation	Result
x or y	if x is false, then y, else x
x and y	if x is false, then x, else y
not x	if x is false, then True, else False

Comparisons

Operation	Meaning
<	strictly less than
<=	less than or equal
>	strictly greater than
>=	greater than or equal
==	equal
!=	not equal
is	object identity if type(1) is int: print(1) >> 1
is not	negated object identity

Numeric Types (int, float, complex) Operations

Operation	Result
x + y	sum of x and y
x - y	difference of x and y
x * y	product of x and y
x / y	quotient of x and y
x // y	floored quotient of x and y
x % y	remainder of x / y
abs(x)	absolute value or magnitude of x
int(x)	x converted to integer
float(x)	x converted to floating point
complex(re, im)	a complex number with real part re, imaginary part im. imdefaults to zero.
divmod(x, y)	the pair (x // y, x % y)
pow(x, y)	x to the power y
x ** y	x to the power y

Getting Help

```
In [33]: help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current
sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

```
In [34]: print?
Docstring:
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file:  a file-like object (stream); defaults to the current sys.stdout.
sep:   string inserted between values, default a space.
end:   string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
Type:      builtin_function_or_method
```

```
In [35]: x=1

In [36]: x?
Type:        int
String form: 1
Docstring:
int([x]) -> integer
int(x, base=10) -> integer
```

```
In [37]: l=[1,2,3]

In [38]: l.
```

```
append
clear
copy
count
extend
index
insert
```

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!*

Master in Software Engineering

Hussam Hourani has over 25 years of Organizations Transformation, VROs, PMO, Large Scale and Enterprise Programs Global Delivery, Leadership, Business Development and Management Consulting. His client experience is wide ranging across many sectors but focuses on Performance Enhancement, Transformation, Enterprise Program Management, Artificial Intelligence and Data Science.