POWERING POTENTIAL

Python Variables

Module 9

FUNDAMENTAL VARIABLES

Contents

- Python objects
- Python variables
- Type specific methods
- Augmented assignments
- Python types
- Python lists
- Python tuples
- Python dictionaries

Summary

- Python operators
- Python reserved words

Python is object oriented

So what is an object?

- To a mathematician the term describes a "thing"
- To a programmer the term describes a specific area of memory

Objects have type, state, and identity

- An object's type is called its *Class* *

→ Describes the size and format of the area of memory

→ Describes the actions which may be carried out on the object

- An object's state is the value of data held in the memory
- An object's identity is its unique position (*address*) in memory

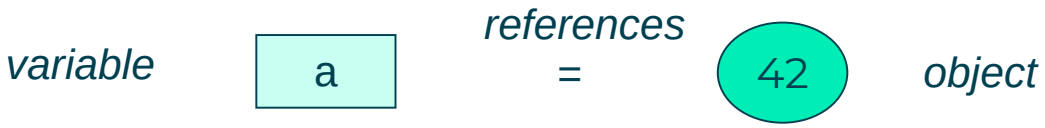
→ Python uses the memory location to identify the object

→ Python does not expose memory addresses directly

If that is an object, what is a variable?

Python variables

- **Python variables are references to objects**



- **Variables are defined automatically**

→ An undefined variable refers to a special object called None

- **Variables can be deleted with `del`**

→ An object's memory can be reused when it is no longer referenced

- **Variables are local by default**

→ If created within a user written function

→ More on `global` variables and scope later…

→ Display local variables with `print(locals())`

Variable names

- **Case sensitive**
- **Usual rules for the name**

→ Must start with an underscore or a letter

→ Followed by any number of letters, digits, or underscores

- **<u>Conventions</u> with underscores**

→ Names beginning with one underscore are private to a module/class

```
_private_to_module
```

→ Names beginning with two underscores are mangled

```
__private_to_class
```

→ Names beginning and ending with two underscores are special

```
__itsakindamagic__
```

→ The character _ represents the result of the previous command

Type specific methods

Actions on objects are done by calling *methods*

- A method is implemented as a *function* - a named code block

object.*method* ([*arg1*[,*arg2*…]])

- *object* need not be a variable

Which methods may be used?

- Depends on the Class (type) of the object
- `dir(`*object*`)` lists the methods available
- `help(`*object*`)` often gives help text

Examples:

```
name.upper()                    names.pop()
name.isupper()                  mydict.keys()
names.count()                   myfile.flush()
```

Operators and type

An operator carries out an operation on an object

- Produces a result which does not (usually) alter the object
- The operation depends on the Class (type) of the object

→ List the Class using the **`type`** built-in function

```
a = 42
b = 9
print(a + b)
print(type(a))

a = 'Hello '
b = 'World!'
print(a + b)
print(type(a))
```

a and b refer to integers

```
51
<class 'int'>
```

a and b now refer to strings

```
Hello World!
<class 'str'>
```

- A list of Python operators is given after the chapter summary

py3

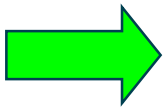
Python 2 reported `type` instead of `class`

Augmented assignments

A convenient shorthand for some assignments

```
stein = 1
pint  = 1
```

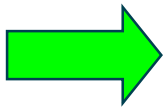
```
stein = stein + pint
```



```
stein += pint
```

Use any arithmetic operator

```
lhs = lhs + rhs
lhs = lhs - rhs
lhs = lhs * rhs
lhs = lhs / rhs
lhs = lhs % rhs
```



```
lhs += rhs
lhs -= rhs
lhs *= rhs
lhs /= rhs
lhs %= rhs
```

Augmented assignment is an assignment!
It has a result, which is usually ignored.

Python 3 types

- Numbers

```
3.142, 42, 0x3f, 0o664
```

Sequences

- Bytes

```
b'Norwegian Blue', b"Mr. Khan's bike"
```

- Strings

```
'Norwegian Blue', "Mr. Khan's bike", r'C:\Numbers'
```

- Tuples

```
(47, 'Spam', 'Major', 683, 'Ovine Aviation')
```

Immutable

- Lists

```
['Cheddar', ['Camembert', 'Brie'], 'Stilton']
```

Mutable

- Bytearrays

```
bytearray(b'abc')
```

- Dictionaries

```
{'Sword':'Excalibur', 'Bird':'Unladen Swallow'}
```

- Sets

```
{'Chapman', 'Cleese', 'Idle', 'Jones', 'Palin'}
```

Switching types

Sometimes Python switches types automatically

```
num = 42
pi  = 3.142
num = 42/pi
print(num)
```

`num` gets automatic promotion

```
13.367281986
```

Sometimes you have to encourage it

- This avoids unexpected changes of type

```
print("Unused port: "  +  count)
TypeError: Can't convert 'int' object to str implicitly
```

- Use the `str()` function to return an object as a string
- Use `int()` or `float()` to return an object as a number
- Other functions available to return lists and tuples from strings

```
print("Unused port: " + str(count))
```

Python lists introduced

Python lists are similar to arrays in other languages

- Items may be accessed from the left by an index starting at 0
- Items may be accessed from the right by an index starting at -1
- Specified as a comma-separated list of objects inside [ ]

```
cheese = ['Cheddar', 'Stilton', 'Cornish Yarg']
print(cheese[1])
cheese[-1] = 'Red Leicester'
print(cheese)
```

```
Stilton
['Cheddar', 'Stilton', 'Red Leicester']
```

Multi-dimensional lists are just lists containing others

```
cheese = ['Cheddar', ['Camembert', 'Brie'], 'Stilton']
print(cheese[1][0])
```

```
Camembert
```

Python tuples introduced

Tuples are *immutable* (read-only) objects

- Specified as a comma-separated list of objects, often inside ( )
→ Can be specified inside ( ) sometimes required for precedence
→ The comma makes a tuple, not the ( )
- Can be indexed in the same way as lists
→ Starting from 0 on the left or -1 on the right

```
mytuple = 'eggs', 'bacon', 'spam', 'tea'
print(mytuple)
print(mytuple[1])
print(mytuple[-1])
```

```
('eggs', 'bacon', 'spam', 'tea')
bacon
tea
```

- Can be reassigned, but not altered

```
mytuple[2] = 'John'
```

```
TypeError: 'tuple' object does not support item assignment
```

Python dictionaries introduced

A Dictionary object is an unordered collection of objects

- Constructed from { }

  varname **=** **{***key1***:***object1***,***key2***:***object2***,***key3***:***object3***,…}**

- Accessed by key

→ A key is a text string, or anything that yields a text string

varname **[***key***]** **=** *object*

```
mydict = {'Australia':'Canberra', 'Eire':'Dublin',
          'France':'Paris', 'Finland':'Helsinki',
          'UK':'London', 'US':'Washington’
         }
print(mydict['UK'])

country = 'Iceland'
mydict[country] = 'Reykjavik'
```

```
London
```

SUMMARY

- **A Python variable is a reference to an object**
- **Python variable names are case-sensitive**
- Watch out for leading underscores
- **Variables are accessed using operators and methods**
- `dir(`*object*`)` lists the methods available
- **Lists are like arrays in other languages**
- **Tuples are "immutable"**
- But can contain variables
- **Dictionaries store objects accessed by key**
- Keys are unique
- Not ordered

Python operators

`or`	logical OR
`and`	logical AND
`not`	logical NOT
`< <= > >=`	comparison operators
`== !=`	equality operators
`is`	object identity test
`in`	object membership test
`|  ^`	binary OR, XOR
`&`	binary AND
`<<  >>`	binary shift
`- +`	subtract, add
`*  / // %`	multiply, divide, integer-divide, modulo
`@`	matrix multiplication (3.5)
`~ **`	complement, exponentiation
`await`	await expression (3.5)

Python reserved words

The following are illegal as variable or function names in Python

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

* version 2.6 and later
\+ version 3.0
~ not in version 3.0
^ version 3.7

exec and **print** were keywords prior to 3.0, now they are built-in functions

py3