Science & Technology Facilities Council | UK Research and Innovation | NERC SCIENCE OF THE ENVIRONMENT

Python

Basics

Centre for Environmental Data Analysis
SCIENCE AND TECHNOLOGY FACILITIES COUNCIL
NATURAL ENVIRONMENT RESEARCH COUNCIL

software carpentry

National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

National Centre for Earth Observation
NATURAL ENVIRONMENT RESEARCH COUNCIL

A simple interpreted language

A simple interpreted language

no separate compilation step

A simple interpreted language

no separate compilation step

```
$ python
>>>
```

A simple interpreted language

no separate compilation step

```
$ python
>>> print(1 + 2)
3
>>>
```

A simple interpreted language

no separate compilation step

```
$ python
>>> print(1 + 2)
3
>>> print('Charles' + 'Darwin')
CharlesDarwin
```

Or remove print (when in the interactive python shell):

```
>>> 'Charles' + 'Darwin'
CharlesDarwin
```

Put commands in a file and execute that

Put commands in a file and execute that

```
$ gedit very-simple.py
```

Put commands in a file and execute that

```
$ gedit very-simple.py
```

```
print(1 + 2)
print('Charles' + 'Darwin')
```

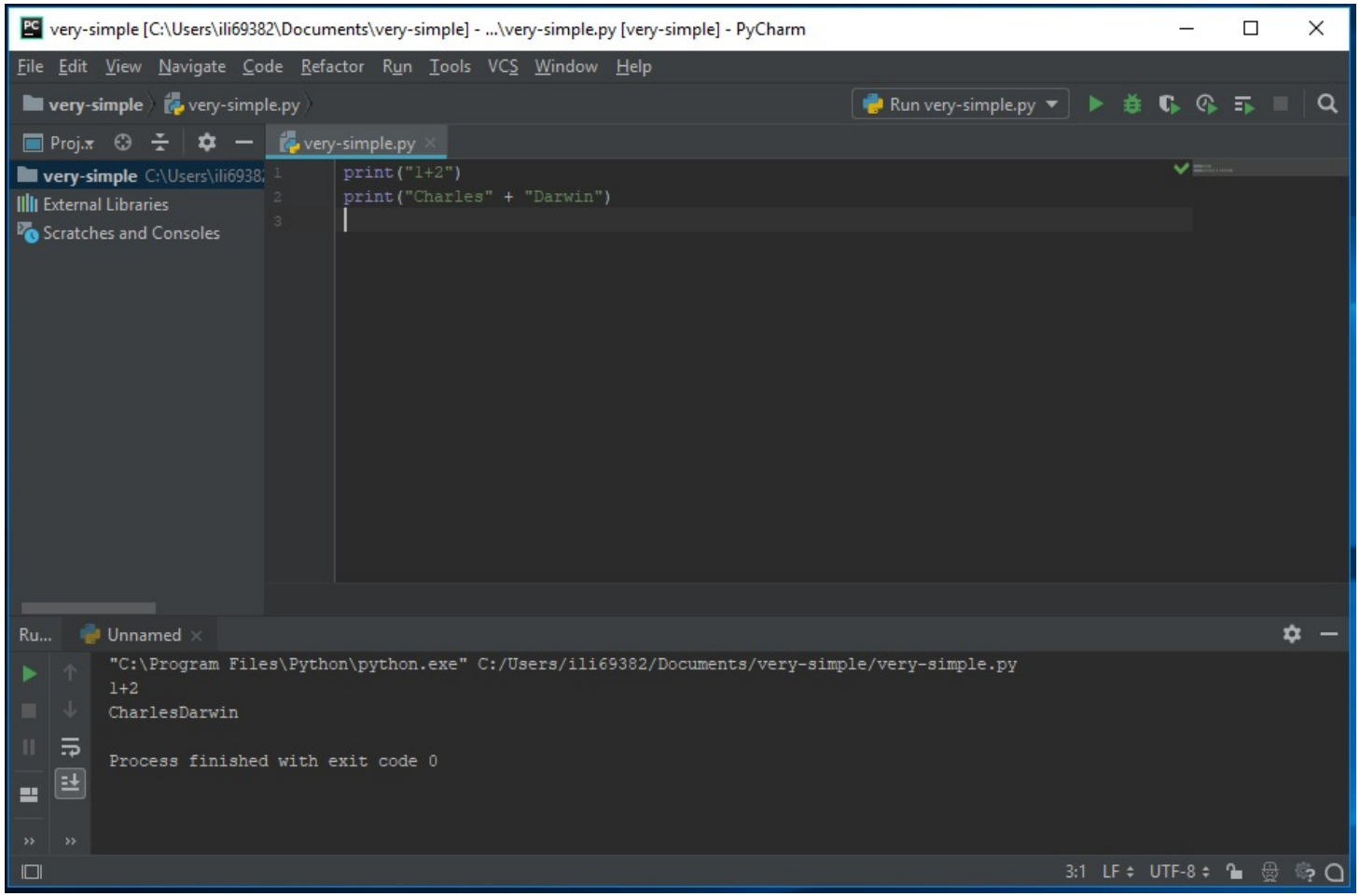
Put commands in a file and execute that

```
$ gedit very-simple.py
```

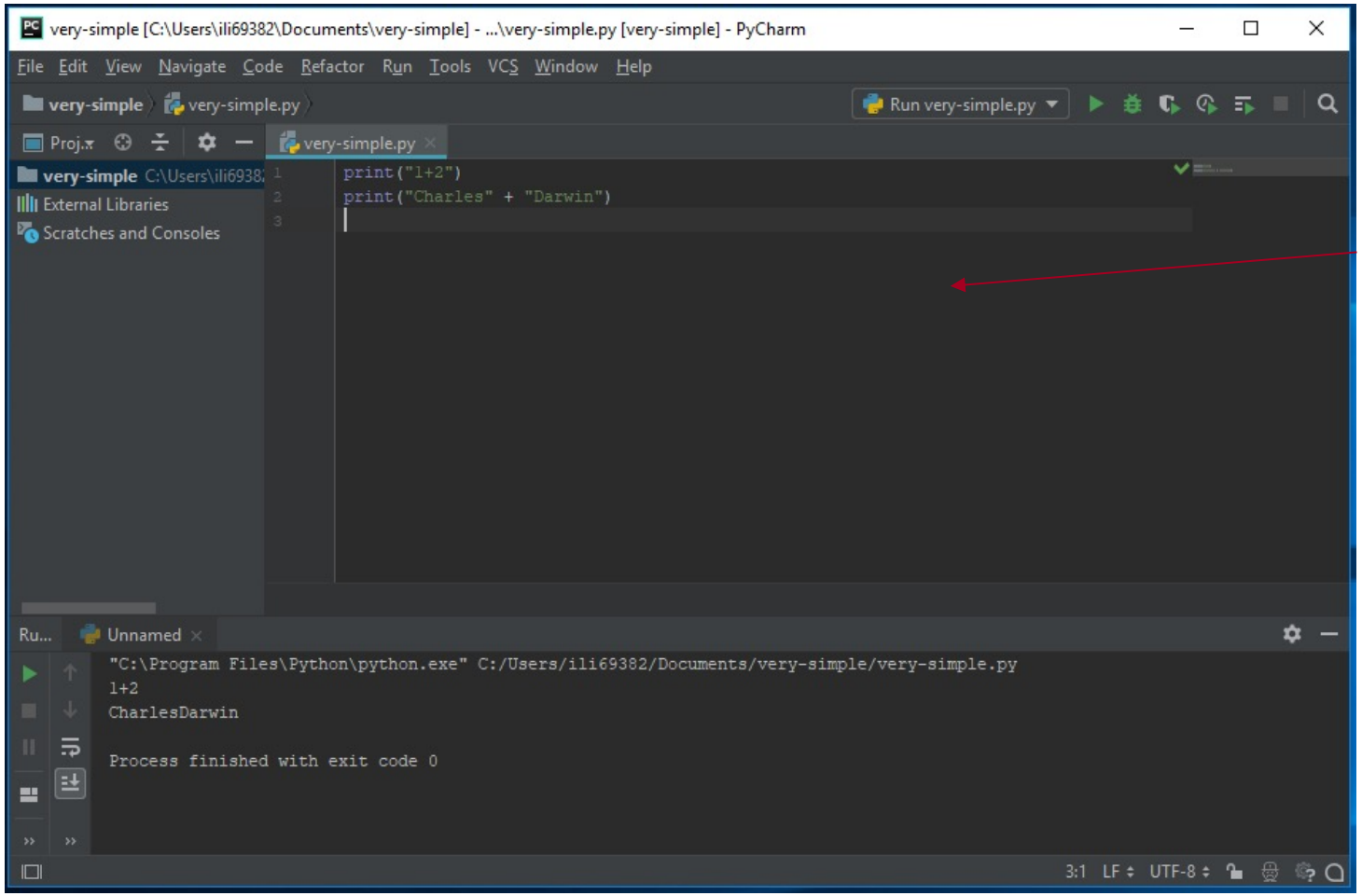
```
print(1 + 2)
print('Charles' + 'Darwin')
```

```
$ python very-simple.py
3
CharlesDarwin
$
```

Use an *integrated development environment* (IDE)

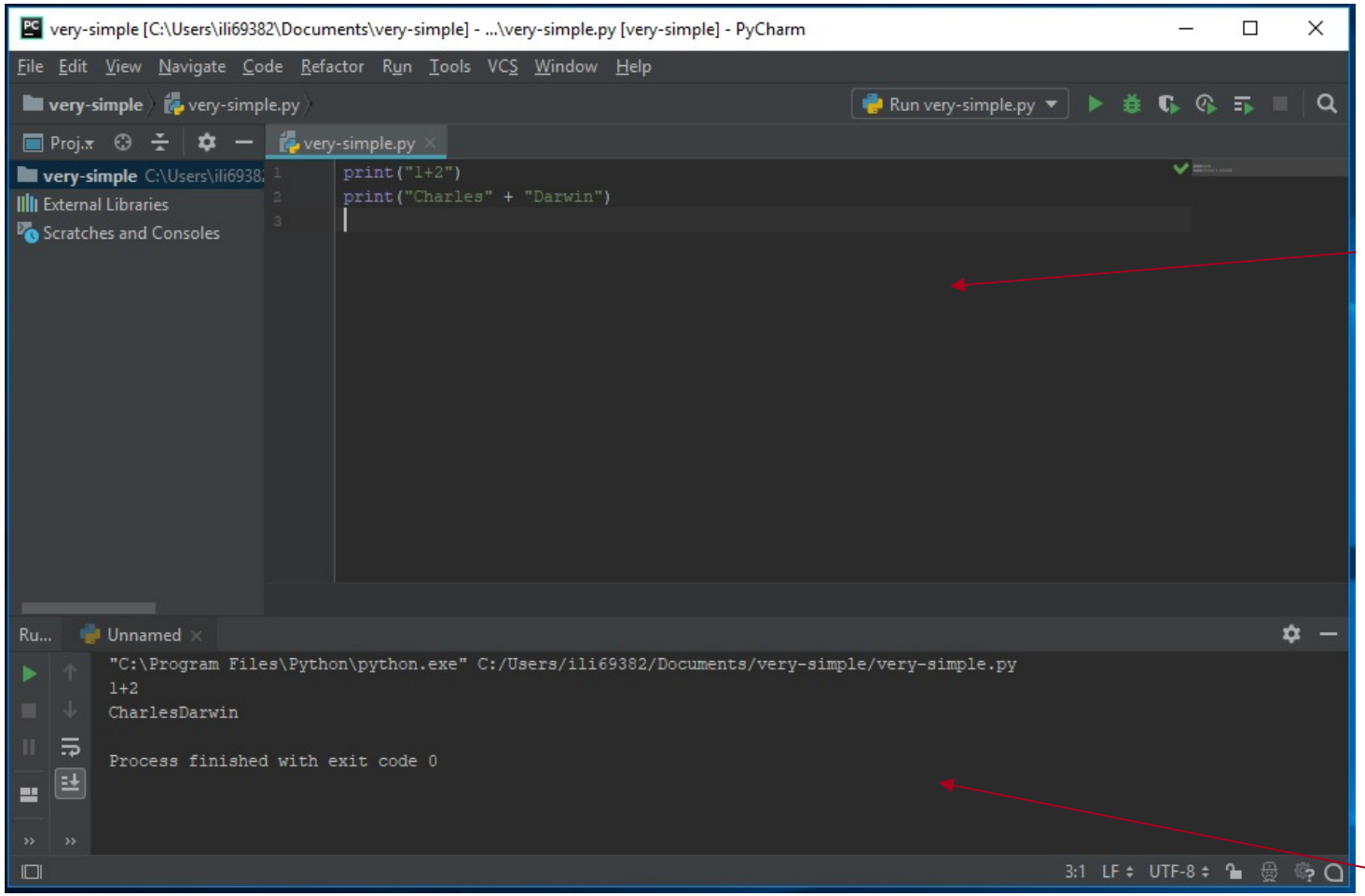


Use an *integrated development environment* (IDE)



Source file

Use an *integrated development environment* (IDE)



Source file

Execution shell

Variables are names for values

Variables are names for values

Created by use

Variables are names for values

Created by use: no declaration necessary

Variables are names for values

Created by use: no declaration necessary

```
>>> planet = 'Pluto'
>>>
```

Variables are names for values

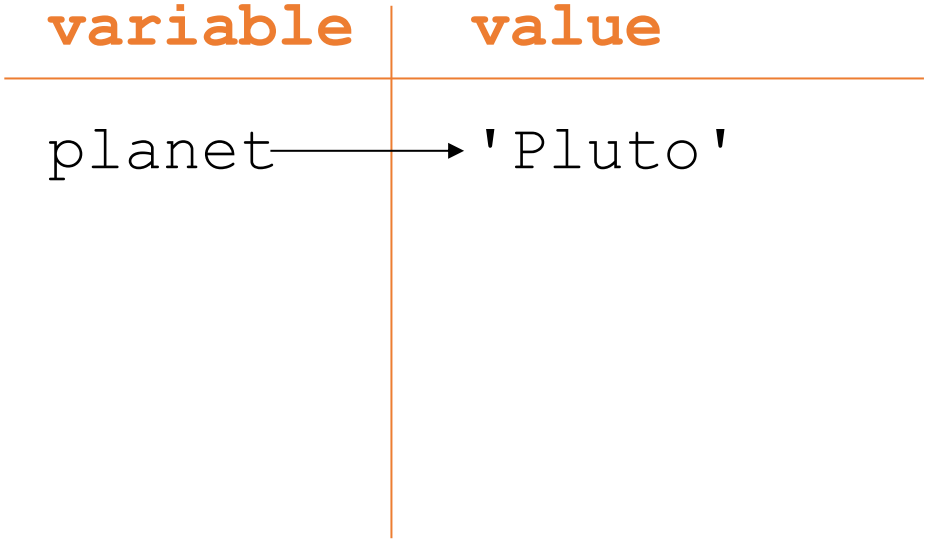
Created by use: no declaration necessary

```
>>> planet = 'Pluto'
>>> print(planet)
Pluto
>>>
```

Variables are names for values

Created by use: no declaration necessary

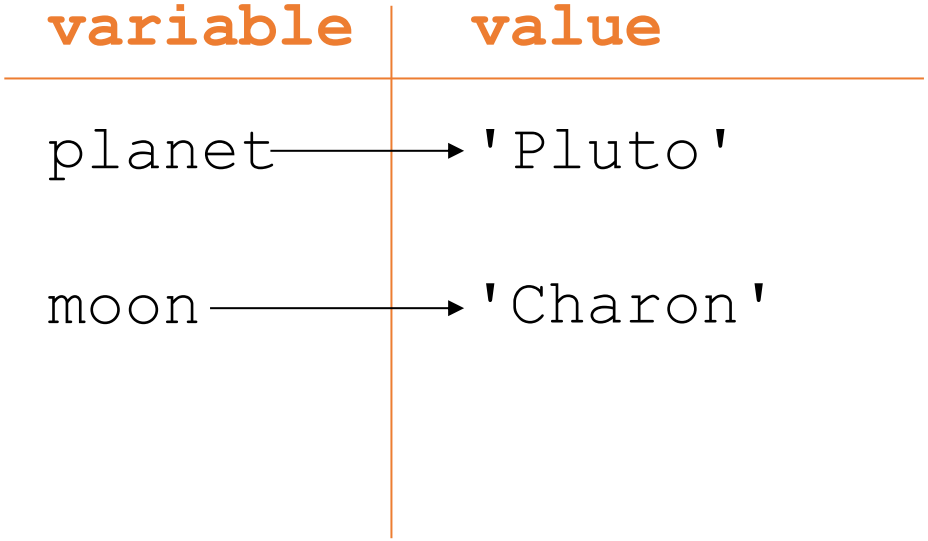
```
>>> planet = 'Pluto'
>>> print(planet)
Pluto
>>>
```



Variables are names for values

Created by use: no declaration necessary

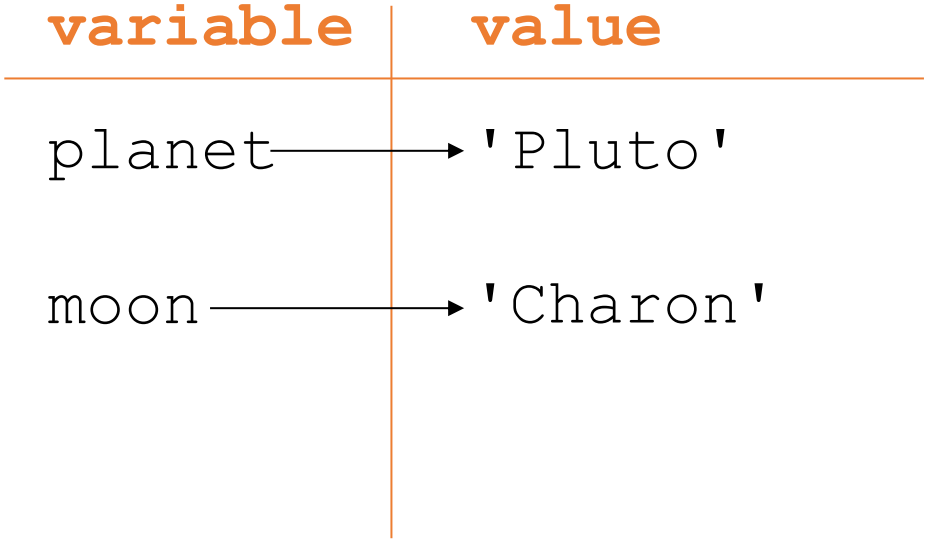
```
>>> planet = 'Pluto'
>>> print(planet)
Pluto
>>> moon = 'Charon'
>>>
```



Variables are names for values

Created by use: no declaration necessary

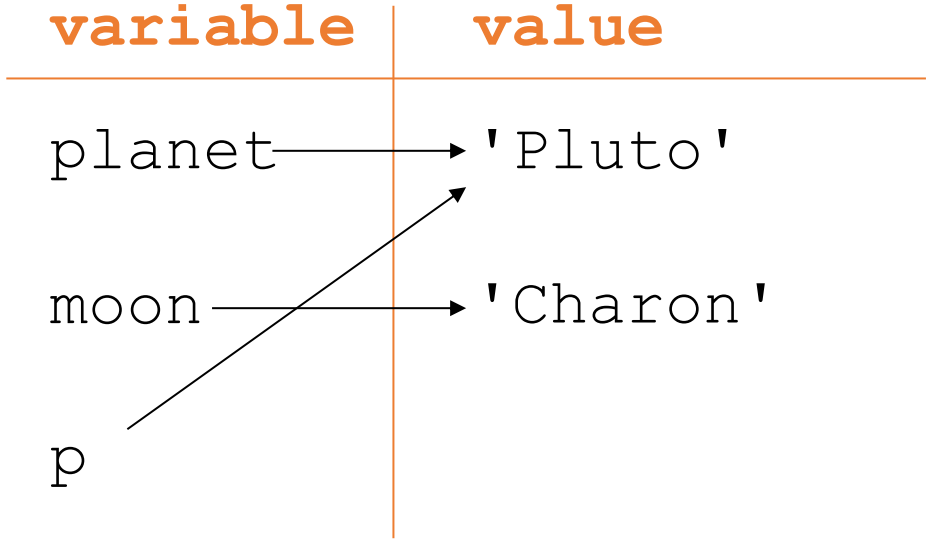
```
>>> planet = 'Pluto'
>>> print(planet)
Pluto
>>> moon = 'Charon'
>>> p = planet
>>>
```



Variables are names for values

Created by use: no declaration necessary

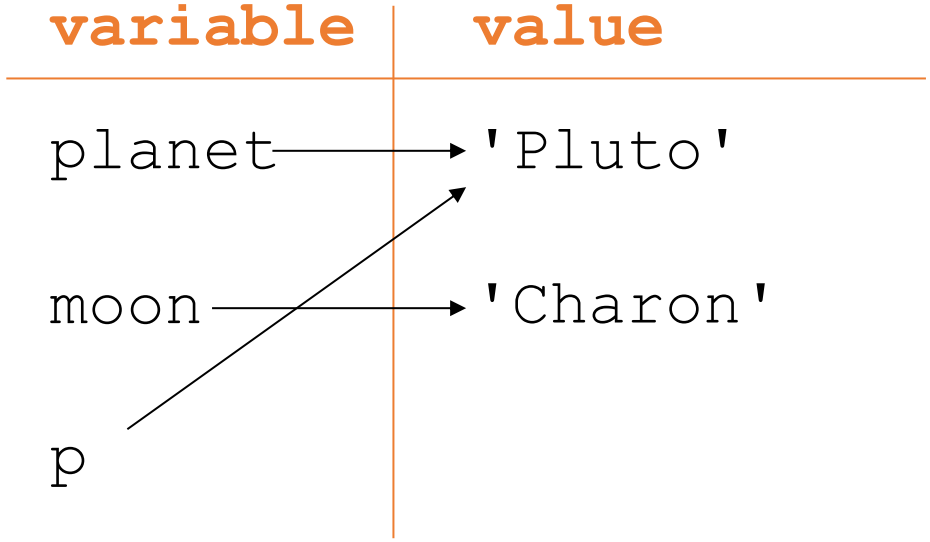
```
>>> planet = 'Pluto'
>>> print(planet)
Pluto
>>> moon = 'Charon'
>>> p = planet
>>>
```



Variables are names for values

Created by use: no declaration necessary

```
>>> planet = 'Pluto'
>>> print(planet)
Pluto
>>> moon = 'Charon'
>>> p = planet
>>> print(p)
Pluto
>>>
```



A variable is just a name

A variable is just a name

Does not have a type

A variable is just a name

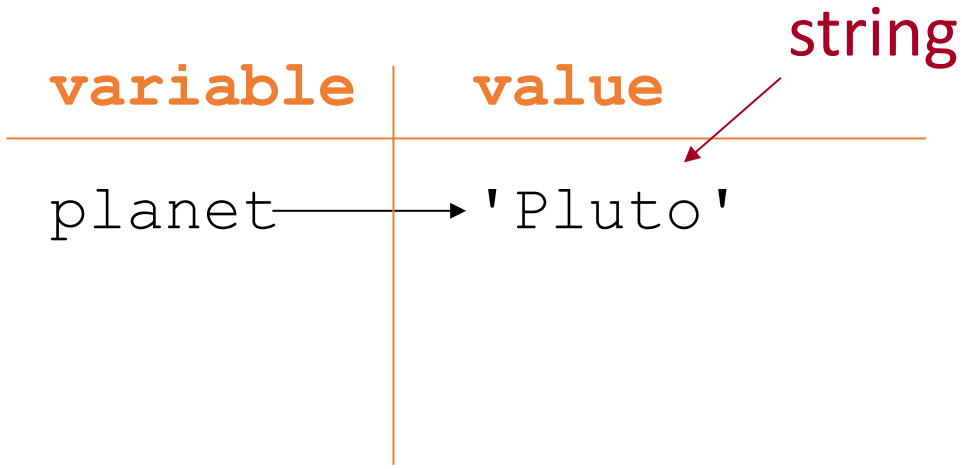
Does not have a type

```
>>> planet = 'Pluto'
>>>
```

A variable is just a name

Does not have a type

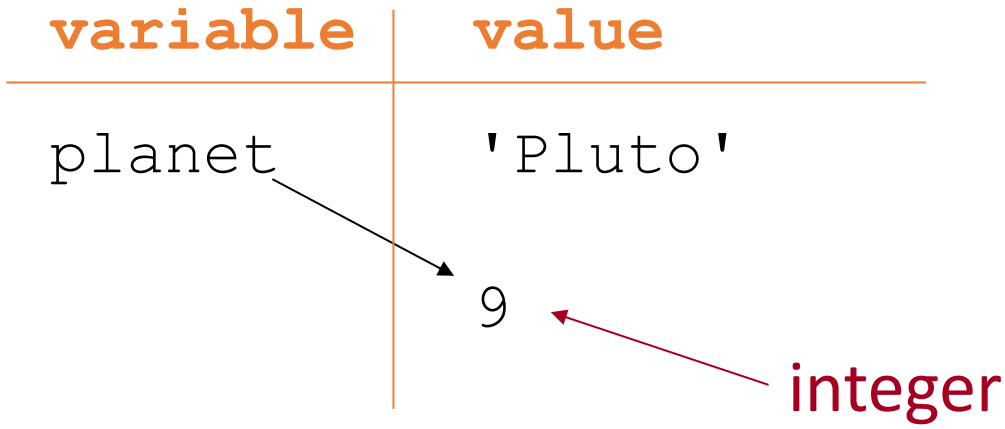
```
>>> planet = 'Pluto'
>>>
```



A variable is just a name

Does not have a type

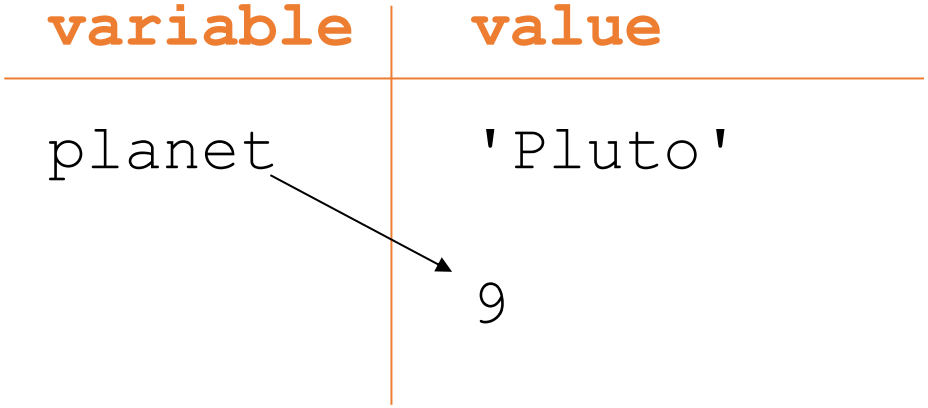
```
>>> planet = 'Pluto'
>>> planet = 9
>>>
```



A variable is just a name

Does not have a type

```
>>> planet = 'Pluto'
>>> planet = 9
>>>
```

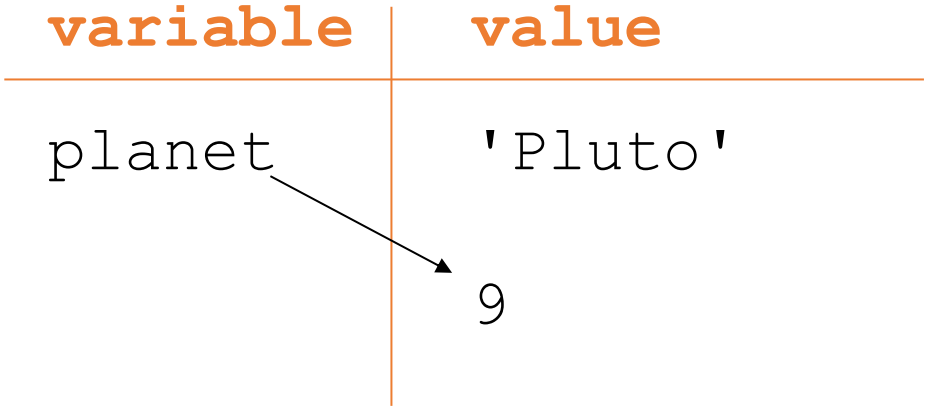


Values are *garbage collected*

A variable is just a name

Does not have a type

```
>>> planet = 'Pluto'
>>> planet = 9
>>>
```



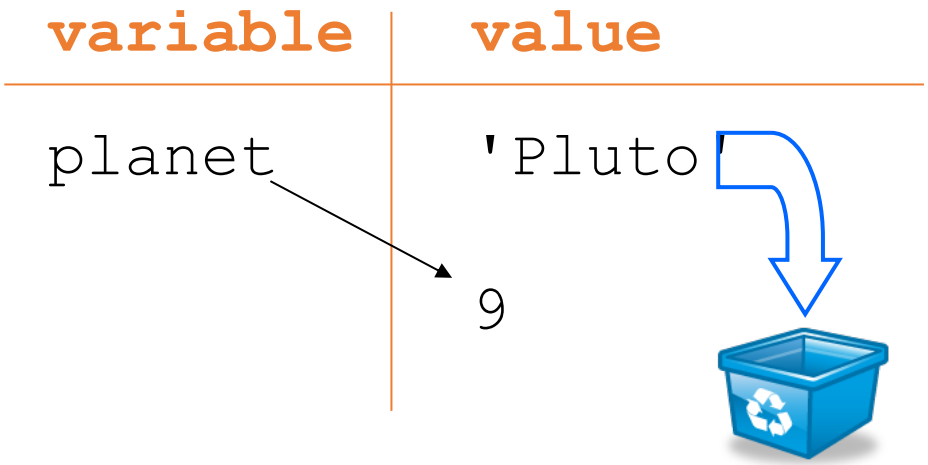
Values are *garbage collected*

If nothing refers to data any longer, it can be recycled

A variable is just a name

Does not have a type

```
>>> planet = 'Pluto'
>>> planet = 9
>>>
```



Values are *garbage collected*

If nothing refers to data any longer, it can be recycled

Must assign value to variable before using it

Must assign value to variable before using it

```
>>> planet = 'Sedna'
>>>
```

Must assign value to variable before using it

```
>>> planet = 'Sedna'
>>> print(plant)# note the deliberate misspelling
```

Must assign value to variable before using it

```
>>> planet = 'Sedna'
>>> print(plant)# note the deliberate misspelling
Traceback (most recent call last):
    print(plant)
NameError: name 'plant' is not defined
>>>
```

Must assign value to variable before using it

```
>>> planet = 'Sedna'
>>> print(plant)# note the deliberate misspelling
Traceback (most recent call last):
    print(plant)
NameError: name 'plant' is not defined
>>>
```

Python does not assume default values for variables

Must assign value to variable before using it

```
>>> planet = 'Sedna'
>>> print(plant)# note the deliberate misspelling
Traceback (most recent call last):
    print(plant)
NameError: name 'plant' is not defined
>>>
```

Python does not assume default values for variables

Doing so can mask many errors

Must assign value to variable before using it

```
>>> planet = 'Sedna'
>>> print(plant)# note the deliberate misspelling
Traceback (most recent call last):
    print(plant)
NameError: name 'plant' is not defined
>>>
```

Python does not assume default values for variables

Doing so can mask many errors

Anything from # to the end of the line is a comment

Values *do* have types

Values *do* have types

```
>>> string = "two"
>>> number = 3
>>> print(string * number)# repeated concatenation
twotwotwo
>>>
```

Values *do* have types

```
>>> string = "two"
>>> number = 3
>>> print(string * number)# repeated concatenation
twotwotwo
>>> print(string + number)
Traceback (most recent call last)
    number + string
TypeError: can only concatenate str
(not "int") to str
>>>
```

Values *do* have types

```
>>> string = "two"
>>> number = 3
>>> print(string * number)# repeated concatenation
twotwotwo
>>> print(string + number)
Traceback (most recent call last)
    number + string
TypeError: can only concatenate str
(not "int") to str
>>>
```

Would probably be safe here to produce `'two3'`

Values *do* have types

```
>>> string = "two"
>>> number = 3
>>> print(string * number)# repeated concatenation
twotwotwo
>>> print(string + number)
Traceback (most recent call last)
    number + string
TypeError: can only concatenate str
(not "int") to str
>>>
```

Would probably be safe here to produce `'two3'`

But then what should `'2'+'3'` be?

Values *do* have types

```
>>> string = "two"
>>> number = 3
>>> print(string * number)# repeated concatenation
twotwotwo
>>> print(string + number)
Traceback (most recent call last)
    number + string
TypeError: can only concatenate str
(not "int") to str
>>>
```

Would probably be safe here to produce `'two3'`

But then what should `'2'+'3'` be?

Doing too much is as bad as doing too little…

Use functions to convert between types

Use functions to convert between types

```
>>> print(int('2') + 3)
5
>>>
```

Use functions to convert between types

```
>>> print(int('2') + 3)
5
>>> print('2' + str(3))
23
>>>
```

Numbers

Numbers

`14`	integer with unlimited precision (as much memory as available)

Numbers

`14`	integer with unlimited precision (as much memory as available)
`14.0`	64-bit float (on most machines)

Numbers

`14`	integer with unlimited precision (as much memory as available)
`14.0`	64-bit float (on most machines)
`1+4j`	complex number (two 64-bit floats)

Numbers

`14`	integer with unlimited precision (as much memory as available)
`14.0`	64-bit float (on most machines)
`1+4j`	complex number (two 64-bit floats)
`x.real`, `x.imag`	real and imaginary parts of complex number

Arithmetic

Arithmetic

Addition	+	35 + 22	57

Arithmetic

Addition	+	`35 + 22`	`57`
		`'Py' + 'thon'`	`'Python'`

Arithmetic

Addition	+	`35 + 22`	`57`
		`'Py' + 'thon'`	`'Python'`
Subtraction	-	`35 - 22`	`13`

Arithmetic

Addition	+	`35 + 22`	`57`
		`'Py' + 'thon'`	`'Python'`
Subtraction	-	`35 - 22`	`13`
Multiplication	*	`3 * 2`	`6`

Arithmetic

Addition	+	`35 + 22`	`57`
		`'Py' + 'thon'`	`'Python'`
Subtraction	-	`35 - 22`	`13`
Multiplication	*	`3 * 2`	`6`
		`'Py' * 2`	`'PyPy'`

Arithmetic

Addition	+	`35 + 22`	`57`
		`'Py' + 'thon'`	`'Python'`
Subtraction	-	`35 - 22`	`13`
Multiplication	*	`3 * 2`	`6`
		`'Py' * 2`	`'PyPy'`
Division	/	`3 / 2`	`1.5`

Arithmetic

Addition	+	`35 + 22`	`57`
		`'Py' + 'thon'`	`'Python'`
Subtraction	-	`35 - 22`	`13`
Multiplication	*	`3 * 2`	`6`
		`'Py' * 2`	`'PyPy'`
Division	/	`3 / 2`	`1.5`
		`3 // 2`	`1`

Arithmetic

Addition	+	`35 + 22`	`57`
		`'Py' + 'thon'`	`'Python'`
Subtraction	-	`35 - 22`	`13`
Multiplication	*	`3 * 2`	`6`
		`'Py' * 2`	`'PyPy'`
Division	/	`3 / 2`	`1.5`
		`3 // 2`	`1`
Exponentiation	**	`2 ** 0.5`	`1.41421356...`

Arithmetic

Addition	+	`35 + 22`	`57`
		`'Py' + 'thon'`	`'Python'`
Subtraction	-	`35 - 22`	`13`
Multiplication	*	`3 * 2`	`6`
		`'Py' * 2`	`'PyPy'`
Division	/	`3 / 2`	`1.5`
		`3 // 2`	`1`
Exponentiation	**	`2 ** 0.5`	`1.41421356...`
Remainder	%	`13 % 5`	`3`

Prefer *in-place* forms of binary operators

Prefer *in-place* forms of binary operators

```
>>> years = 500
>>>
```

Prefer *in-place* forms of binary operators

```
>>> years = 500
>>> years += 1
>>>
```

Prefer *in-place* forms of binary operators

```
>>> years = 500
>>> years += 1
>>>
```

The same as `years = years + 1`

Prefer *in-place* forms of binary operators

```
>>> years = 500
>>> years += 1
>>> print(years)
501
>>>
```

Prefer *in-place* forms of binary operators

```
>>> years = 500
>>> years += 1
>>> print(years)
501
>>> years %= 10
>>>
```

Prefer *in-place* forms of binary operators

```
>>> years = 500
>>> years += 1
>>> print(years)
501
>>> years %= 10
>>>
```

← The same as: `years = years % 10`

Prefer *in-place* forms of binary operators

```
>>> years = 500
>>> years += 1
>>> print(years)
501
>>> years %= 10
>>> print(years)
1
>>>
```

Comparisons

Comparisons

```
3 < 5          | True
```

Comparisons

`3 < 5`	`True`
`3 != 5`	`True`

Comparisons

`3 < 5`	`True`
`3 != 5`	`True`
`3 == 5`	`False`

Comparisons

`3 < 5`	`True`
`3 != 5`	`True`
`3 == 5`	`False`

Single = is assignment

Double == is equality

Comparisons

`3 < 5`	`True`
`3 != 5`	`True`
`3 == 5`	`False`
`3 >= 5`	`False`

Comparisons

`3 < 5`	`True`
`3 != 5`	`True`
`3 == 5`	`False`
`3 >= 5`	`False`
`1 < 3 < 5`	`True`

Comparisons

`3 < 5`	`True`
`3 != 5`	`True`
`3 == 5`	`False`
`3 >= 5`	`False`
`1 < 3 < 5`	`True`
`1 < 5 > 3`	`True`

But please don't do this

Comparisons

`3 < 5`	`True`
`3 != 5`	`True`
`3 == 5`	`False`
`3 >= 5`	`False`
`1 < 3 < 5`	`True`
`1 < 5 > 3`	`True`
`3+2j < 5`	***error***

Python

Boolean types

What is a Boolean?

A simple data type in Python is the Boolean. It has only two possible values:

```
True
False
```

Comparisons result in Boolean values:

```
(3 > 5) is False
(2 == 2.0) is True
```

Tests in if/while result in a Boolean

```
if x > 3:    # Will run if (x > 3) is True
  ...do something...

while my_list:    # Will loop until my_list is empty
   ...something that reduces size of my_list...
```

Note: Boolean values are case-sensitive!

```
True      # but not "true"
False     # but not "false"
```

Science & Technology Facilities Council | UK Research and Innovation | NERC SCIENCE OF THE ENVIRONMENT

Python

Saving your code to a script

Centre for Environmental Data Analysis
SCIENCE AND TECHNOLOGY FACILITIES COUNCIL
NATURAL ENVIRONMENT RESEARCH COUNCIL

software carpentry

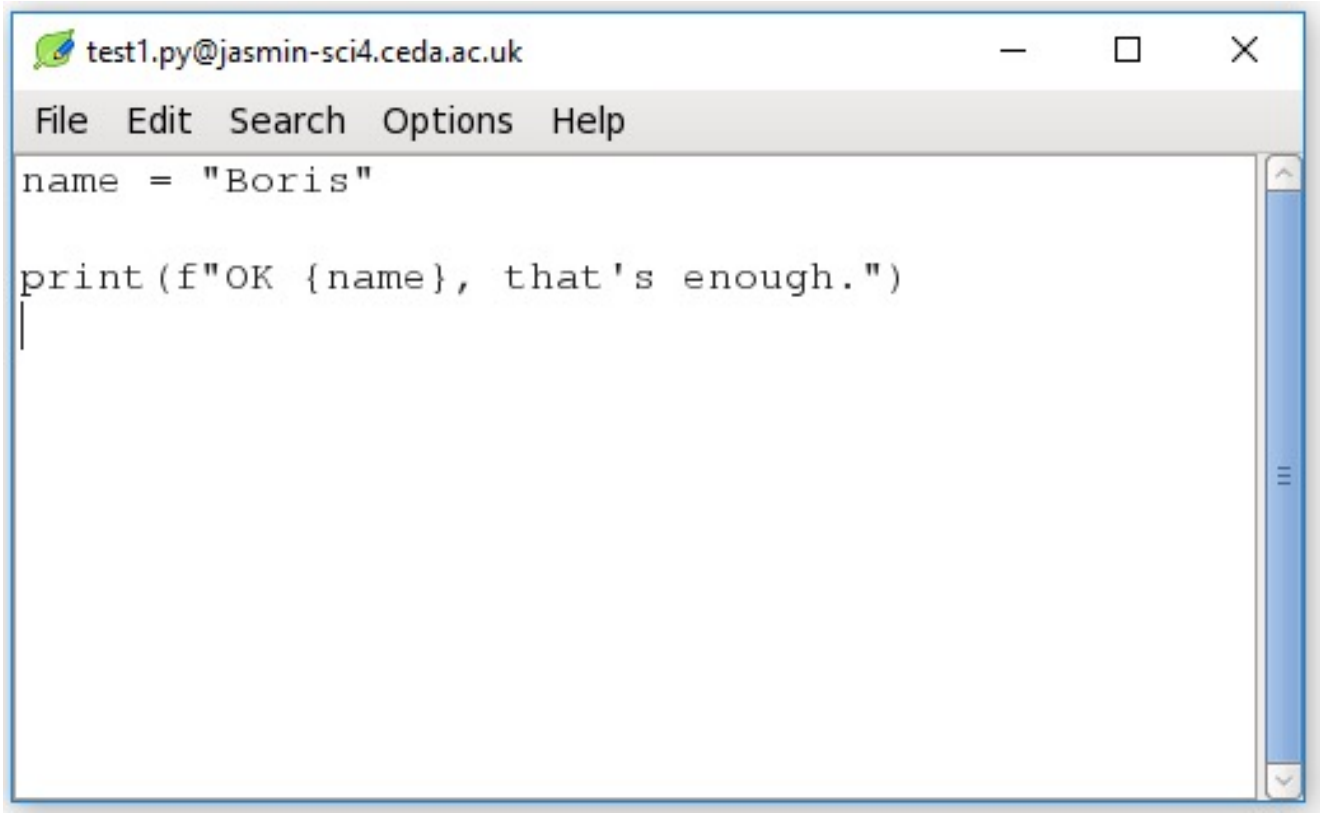
National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

National Centre for Earth Observation
NATURAL ENVIRONMENT RESEARCH COUNCIL

Open an editor (leafpad)

```
$ leafpad test1.py & # "&" means run in background
                     so you can still type here.
```

Opens an editor window... make a change... and **Save!**



Now run it

With...

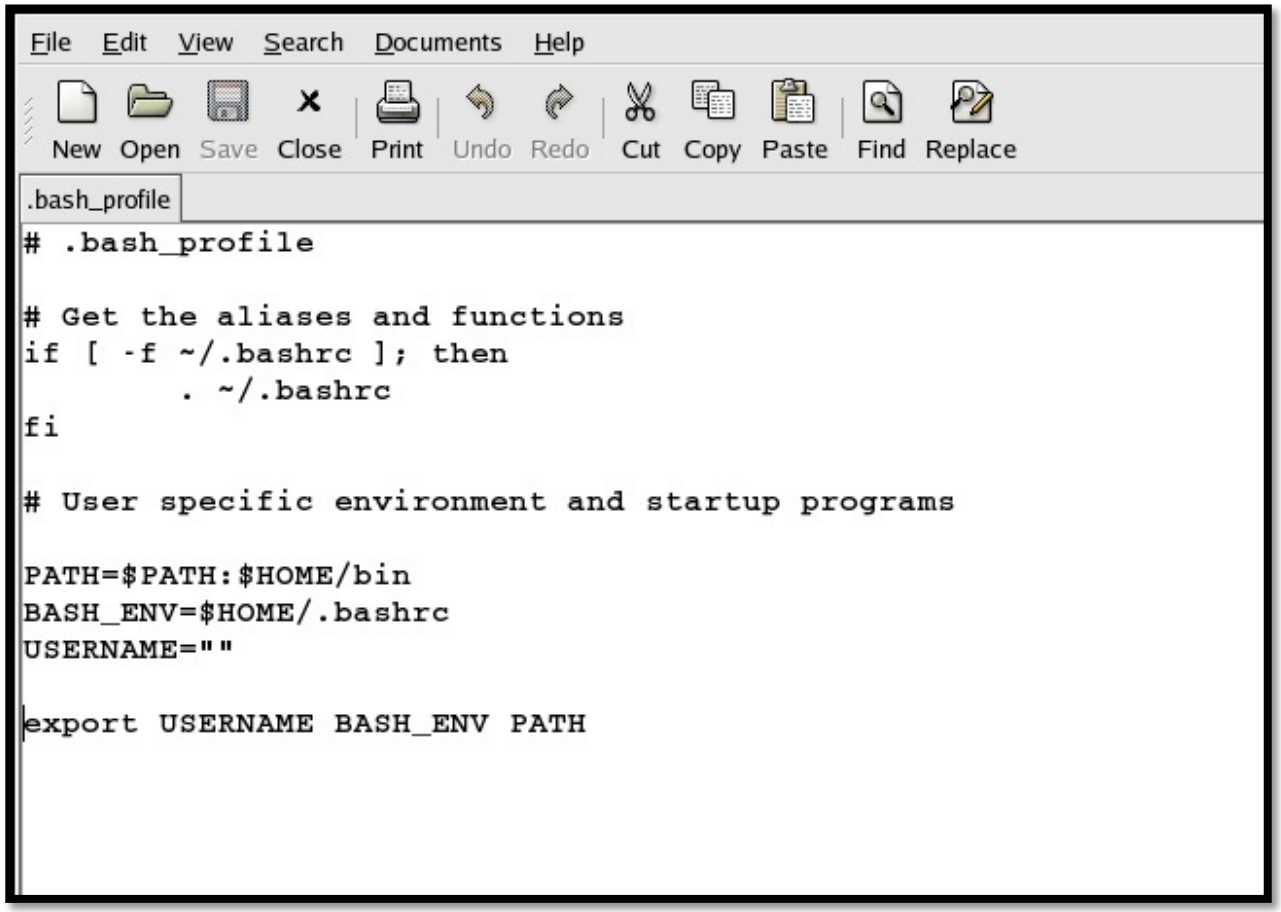
```
$ python test1.py
```

...your output appears here...

Open an editor (gedit)

```
$ gedit .bash_profile &
# "&" means run in background so you can still type here.
```

Opens an editor window... make a change... and **Save!**



created by

Greg Wilson

October 2010