MAKE
SCHOOL

PYTHON

¡Llamas!

CHARACTERISTICS

Multi-purpose scripting language.

Dynamic

No type declarations anywhere.

Interpreted

No separate compile step.

```
$ python
>>> name = "World"
>>> print("Hello, {}!".format(name))
Hello, World!
```

USES

Data analysis

Scientific modeling and computation

Web application development

Natural language processing

...whatever else you want!

QUIRKS

Whitespace affects code structure.

Blocks of statements are grouped by their indentation.

```
def fact(n):
    if n < 1:
        return 0
    if n == 1:
        return n
    return n * fact(n-1)
```

TYPES

```
$ python
>>> # STRINGS
... #
... # strings can be single or double quoted
... single = 'foo'
>>> double = "bar"
>>> # concatenate with + operator
... single + double
'foobar'
>>> # format strings with .format()
... 'name: {}, age: {}'.format('bob', 31)
'name: bob, age: 31'
>>> # use positional arguments
... 'name: {1}, age: {0}'.format(82, 'ann')
'name: ann, age: 82'
>>>
```

```
>>> # get characters by index
... str = 'my string'
>>> str[4]
't'
>>> # slice chunks of a string
... str[3:]
'string'
>>> str[:5]
'my st'
>>> # special characters start with a \
... newline_tab = '\n\t'
>>> # escaping quote characters
... bad_string = 'i'm not here
SyntaxError: invalid syntax
>>> good_string = 'i\'m here!'
>>> good_string
"i'm here!"
```

```
>>> # LISTS
... #
... # one of Python's compound data types
... # mutable, ordered, can contain multiple types
... fibs = [0, 1, 1, 2, 3, 5]
>>> # index starts at 0
... fibs[0]
0
>>> # count backwards using neg numbers
... fibs[-1]
5
>>> # same syntax as strings
... fibs[3:5]
[2, 3]
>>> # concatenate two lists
... fibs + [8, 13]
[0, 1, 1, 2, 3, 5, 8, 13]
```

```
>>> # append single values
... fibs.append(8)
>>> fibs
[0, 1, 1, 2, 3, 5, 8]
>>> # assign values to index
... fibs[2] = 'one'
>>> fibs
[0, 1, 'one', 2, 3, 5, 8]
>>> # assign to slices of a list
... fibs[3:5] = ['two', 'three']
>>> fibs
[0, 1, 'one', 'two', 'three', 5, 8]
>>> # get the length of a list
... len(fibs)
7
>>> # get the index of an element
... fibs.index('three')
4
```

```
>>> # TUPLES
... #
... # another compound, sequence type (like lists)
... # except tuples are _immutable_
... comp = ('Apple', 2012, 'MacBook')
>>> len(comp)
3
>>> comp[0]
'Apple'
>>> # immutable means you can't change an element
... comp[2] = 'iMac'
TypeError: 'tuple' object does not support item
assignment
>>> # but you can nest mutable items inside tuples
... piece = ('Rook', ['b', '7'])
>>> piece[1][0] = 'e'
>>> piece
('Rook', ['e', '7'])
```

```
>>> # tuples can be _unpacked_
... comp = ('Apple', 2012, 'MacBook')
>>> make, year, model = comp
>>> make
'Apple'
>>> year
2012
>>> model
'MacBook'
>>> # and also packed (note lack of parentheses)
... position = ['e', 7]
>>> piece = 'Rook', position
>>> piece
('Rook', ['e', 7])
>>>
```

```
>>> # DICTIONARIES
... #
... # a mapping type consisting of key-value pairs
... # similar to associative arrays in other languages
... card = { 'suit': 'hearts', 'rank': 7 }
>>> # access values by their key
... card['suit']
'hearts'
>>> # add new key-value pairs
... card['played'] = True
>>> # change values with assignment to key
... # (keys are always unique, so this will re-assign)
... card['rank'] = 'K'
>>> card
{'suit': 'hearts', 'rank': 'K', 'played': True}
>>> card.keys()
dict_keys(['suit', 'rank', 'played'])
```

FUNCTIONS

```
>>> # FUNCTIONS
... #
... # named subroutines with zero or more parameters
... # create with 'def func_name(param1, param2):'
... def square(x):
...     return x ** 2
...
>>> # call a function with ()
... square(4)
16
>>> # referencing without () returns a function object
... square
<function square at 0x1089d87b8>
>>> # since they are objects, can assign to variables
... other_square = square
>>> other_square(5)
25
```

```
>>> # functions have access to other functions and
... #  variables defined in their same scope
... secret = 'macandcheese'
>>> def decrypt():
...     print(secret)
...
>>> decrypt()
macandcheese
>>> # functions create a new scope, so variables defined
... #  in a function body are local to that function
... def private():
...     message = 'I like raisins.'
...     print(message)
...
>>> private()
I like raisins.
>>> print(message)
NameError: name 'message' is not defined
```

MODULES AND FILES

MODULES & FILES

Python programs are saved in files with a **.py** extension.

To access code in other files you import them.

Say we have two files in the same directory:

```
./triangle.py
./main.py
```

triangle.py

```
def area(w, h):
    return (w * h) / 2
```

main.py

```
# import triangle _module_
#  from file triangle.py
import triangle

triangle.area(5, 8)
```

THE STANDARD LIBRARY

THE STANDARD LIBRARY

Python comes with a large library of built-in modules.

You can also install third-party libraries.

One of these built-in modules is **`random`**, used to generate random numbers.

main.py

```
import random

# random float
#  between 0 and 1
random.random()
# => 0.8900320849293105

# whole integer
#  within a range
random.randint(10, 20)
# => 15
```

THE STANDARD LIBRARY

Another module in the standard library is the **`pickle`** module.

Use this module to make all kinds of apps, like kim chi, radishes, sauerkraut...

main.py

```
import pickle

# you're going to have to
#  figure this one out on
#  your own...
```

ATTRIBUTION & RESOURCES

https://docs.python.org/3/tutorial/

https://developers.google.com/edu/python/introduction

MAKE SCHOOL

github.com/makeschool/Slides

MAKE
SCHOOL