Tutoriel python de Derek Banas

Here are some notes from a video about python from Derek Banas.

I skipped parts that weren't interesting for me.

Max size

```
sys.maxsize  # get the maximum size of an integer
sys.float_info.max  # get the maximum size of a float
```

Float are accurate up to 15 digits.

Complex numbers : real part + imaginary : `cn1 = 4 + 3j`

Common maths functions

- `abs(float)`
- `max(float, float)`
- `min(float, float)`
- `pow(float, float)  # power`
- `round(float)`

With `math` :

- `math.degrees(float)  # where float is in radian`
- `math.radians(float)  # where float is in degrees`
- `math.hypoth(float, float1)  # =sqrt(float+float1)`
- `math.pi`

- `math.{sin,cos,tan,asin,acos,atan,sinh,cosh,tanh,asinh,acosh,atanh}(float)`
- `math.sqrt(float)`
- `math.log(float, base)`
- `math.exp(float)`
- `math.floor(float)`
- `math.ceil(float)`
- `math.inf`

Random values

Get a random integer in a range :

`random.randint(from, to)`

comparisons

`and` : both are true

`or` : on or both are true

`not` : switch the truth value

Ternary operator

Ternary operators, also known as conditional expressions are operators that evaluate something based on a condition being true or false. (source)

```
canFuck = True if age >= 18 else False  # But I can ;)
```

Strings

Raw string :

```
print(r"There is no escape \n")
```

The \ will not be able to allow characters to escape

You can combine strings :

```
print("Hello " + "You")
```

You can get a tab character with `\t`

```
str = '''Triple quoted "don't need escape" '''
```

`ord()` to get the number of a letter

```
print(19, 1, 1838, sep='/')  # ZEPPELI
```

```
print("No Newline", end='')
```

```
print("%04d %s %.2f %c %e" % (1, "Test", 3.141, 'A', 420000))
```

output : `0001 Test 3.14 A`, ou can use `%e` for exponent

Indexable

`str[from:to:step]`

Test if a word is in a string : `"word" in str`

Get the index of a match : `str.find("word")`

Imutables

```
str[n]="X" # Error
```

use :

```
str = str.replace("Hello", "Goodbye")
```

or

```
str = str[:to] + "y" + str[from:]
```

here nothing implicitly mean everything

Remove trailing and leading whitespace with `str.strip()` `lstrip()` and `rstrip()` are also available

Convert a list into a string with `str.join(list)`, `str` will separate those elements in the new string

Create a list from a string with `str.split(str1)`, `str1` define where to split

f-string :

```
print(f'{int1} + {int2} = {int1 + int2}')
```

Case

- `str.upper()`
- `str.lower()`

Checks

- `str.isalnum()  # char or number`
- `str.isalpha()  # char`
- `str.isdigit()  # num`

Lists

```
l1 = [1, 3.1, "za warudo!", True]
```

Indexable

Mutable

- `list.remove(dio)`
- `list.pop(index)`

Search

- `any in list  # where `any` is of any type`
- `min(list)`
- `max(list)`

Iterable

`iter(object)` where object is iterable return an iterator

This iterator can be cycled through using `next()`

Range

```
l = list(range(start, stop, step))
```

functions

You can make a list out of functions and pass `arg` to them : `list[n](arg)`

Tuples

Tuples are immutable lists

Dictionaries

Create dictionaries from a list of tuples :

```
dictionary = dict([(key, value), (key1, value1)])
```

the inverse function is `list(dictionary.items())` `.keys()` and `.values()` are also available

`del dict[key]` is the same as `dict.pop(key)`

trick to merge two dictionaries :

```
dict2 = {**dict, **dict1}
```

`dict1` will overwrite dict if they conflict

Iterable

Set

A set is an unordered list whose element are unique and immutable while the set itself can change

create a set :

```
s = {"msdlkfj", 1}
```

create a set from a list :

```
s = set(list)
```

merge two set :

```
s2 = s | s1,  # s |= s is available
```

- `s.add("idk")`
- `s.discard("smt")`
- `s.pop()  # random deletion`
- `s.intersection(s1)`
- `s.symmetric_difference(s1) # The opposite of the intersection, uniques values`
- `s.difference(s1)  # in s but not in s1`
- `s.clear()  # delete everything`
- `frozenset(list)  # all of the above is now impossible`

functions

put `*args` in a function arguments definition if you don't know what argument the function will have or how much

`args` is iterable !

Anonymous function

Pass `arg1` to an unnamed function inside a function :

```
function(arg)(arg1)
```

`map()`

`map(function, iterable)`

to compute a function with arguments from each of the iterables

`filter()`

`filter(function, iterable)`

only returns elements from the iterable that return true in the function

`reduce()`

`reduce(function, iterable)`

adds up the iterable after the function is applied to them

Class and objects

Objects have properties and methods, they are instances of a class.

The purposes of class are to create object, they are blueprints for those objects.

Example :

```
class Square:
    # init is used to set values for each Square
    def __init__(self, height="0", width="0"):
        self.height = height
        self.width = width

    # This is the getter
    # self is used to refer to an object that
    # we don't possess a name for
    @property
    def height(self):
        print("Retrieving the height")

        # Put a __ before this private field
        return self.__height

    # This is the setter
    @height.setter
```

```
    def height(self, value):

        # We protect the height from receiving
        # a bad value
        if value.isdigit():

            # Put a __ before this private field
            self.__height = value
        else:
            print("Please only enter numbers for height")

    # This is the getter
    @property
    def width(self):
        print("Retrieving the width")
        return self.__width

    # This is the setter
    @width.setter
    def width(self, value):
        if value.isdigit():
            self.__width = value
        else:
            print("Please only enter numbers for width")

    def get_area(self):
        return int(self.__width) * int(self.__height)

# Create a Square object
square = Square()
square.height = "10"
square.width = "10"
print("Area", square.get_area())

# When a class inherits from another it gets all
# its fields and methods and can change as needed
# See below for more on inheritance and polymorphism
class Animal:
    def __init__(self, name="unknown", weight=0):
        self.__name = name
        self.__weight = weight

    @property
    def name(self, name):
```

```
        self.__name = name

    def make_noise(self):
        return "Grrrrr"

    # Used to cast to a string type
    def __str__(self):
        return "{} is a {} and says {}".format (
        self.__name, type(self).__name__, self.make_noise()
        )

    # Here I'll define how to evaluate greater
    # than between 2 Animal objects using a magic
    # method, see below
    def __gt__(self, animal2):
        if self.__weight > animal2.__weight:
            return True
        else:
            return False


# Dog inherits everything from Animal
class Dog(Animal):
    def __int__(self, name="unknown", owner="unknown", weight=0):
        # Have the super class handle initializing
        Animal.__int__(self, name, weight)
        self.__owner = owner

    # Overwrite str
    def __str__(self):
        # How to call super class methods
        return super().__str__() + " and is owned by " + \
        self.__owner

animal = Animal("Spot", 100)
print(animal)

dog = Dog("Bowser", "Bob", 150)
print(dog)

# Test the magic method
print(animal > dog)
```

Inheritence and Polymorphism

Polymorphism in Python works differently from other languages in that functions accept any object and expect that object to provide the needed method

If you call on a method for an object the method just needs to exist for that object to work.

Magic methods

Magic methods are used for operator overloading

- `__init__`: you already know about that
- `__eq__`: equal
- `__ne__`: not equal
- `__lt__`: less than
- `__gt__`: greater than
- `__le__`: less than or equal
- `__ge__`: greater than or equal
- `__add__`: addition
- `__sub__`: subtraction
- `__mul__`: multiplication
- `__div__`: division
- `__mod__`: modulus

and many others, see https://rszalski.github.io/magicmethods/

Keywords

Keyword	Description
and	A logical operator
as	To create an alias
assert	For debugging
break	To break out of a loop
class	To define a class
continue	To continue to the next iteration of a loop
def	To define a function
del	To delete an object
elif	Used in conditional statements, same as else if
else	Used in conditional statements

Keyword	Description
except	Used with exceptions, what to do when an exception occurs
False	Boolean value, result of comparison operations
finally	Used with exceptions, a block of code that will be executed no matter if there is an exception or not
for	To create a for loop
from	To import specific parts of a module
global	To declare a global variable
if	To make a conditional statement
import	To import a module
in	To check if a value is present in a list, tuple, etc.
is	To test if two variables are equal
lambda	To create an anonymous function
None	Represents a null value
nonlocal	To declare a non-local variable
not	A logical operator
or	A logical operator
pass	A null statement, a statement that will do nothing
raise	To raise an exception
return	To exit a function and return a value
True	Boolean value, result of comparison operations
try	To make a try…except statement
while	To create a while loop
with	Used to simplify exception handling
yield	To end a function, returns a generator

Source

Voir le PDF