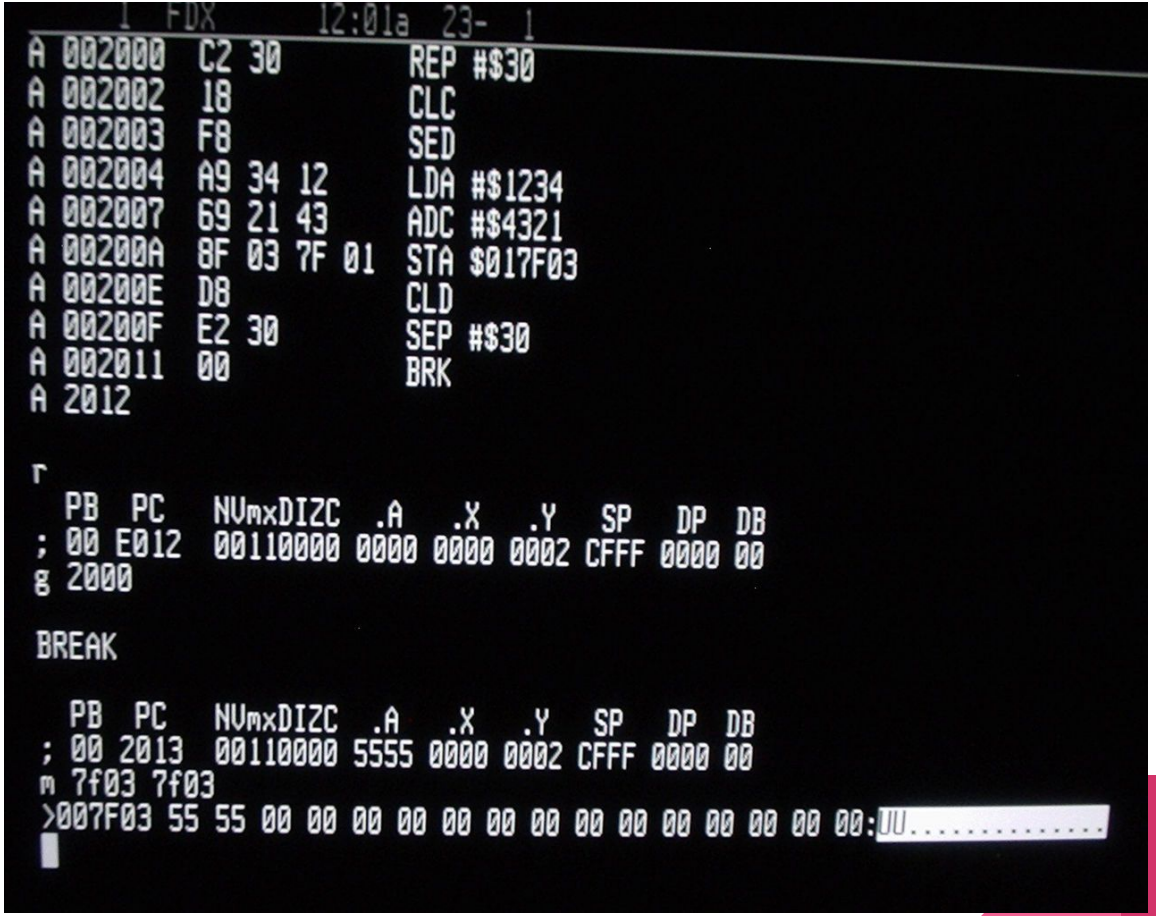
Python Paradigms

Petr Svarny, 2020

How do you code?



https://commons.wikimedia.org/wiki/File:W65C816S_Machine_Code_Monitor.jpeg#/media/File:W65C816S_Machine_Code_Monitor.jpeg

Assembler

Try out:

[Tutorialspoint]

[TIS-100]

```
MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER  PAGE    2


C000                    ORG     ROM+$0000 BEGIN MONITOR
C000 8E 00 70  START    LDS     #STACK

                        ****************************************
                        * FUNCTION: INITA - Initialize ACIA
                        * INPUT: none
                        * OUTPUT: none
                        * CALLS: none
                        * DESTROYS: acc A

0013                    RESETA  EQU     %00010011
0011                    CTLREG  EQU     %00010001

C003 86 13      INITA   LDA A   #RESETA   RESET ACIA
C005 B7 80 04           STA A   ACIA
C008 86 11              LDA A   #CTLREG   SET 8 BITS AND 2 STOP
C00A B7 80 04           STA A   ACIA

C00D 7E C0 F1           JMP     SIGNON    GO TO START OF MONITOR

                        ****************************************
                        * FUNCTION: INCH - Input character
                        * INPUT: none
                        * OUTPUT: char in acc A
                        * DESTROYS: acc A
                        * CALLS: none
                        * DESCRIPTION: Gets 1 character from terminal

C010 B6 80 04   INCH    LDA A   ACIA      GET STATUS
C013 47                 ASR A             SHIFT RDRF FLAG INTO CARRY
C014 24 FA              BCC     INCH      RECIEVE NOT READY
C016 B6 80 05           LDA A   ACIA+1    GET CHAR
C019 84 7F              AND A   #$7F      MASK PARITY
C01B 7E C0 79           JMP     OUTCH     ECHO & RTS

                        ****************************************
                        * FUNCTION: INHEX - INPUT HEX DIGIT
                        * INPUT: none
                        * OUTPUT: Digit in acc A
                        * CALLS: INCH
                        * DESTROYS: acc A
                        * Returns to monitor if not HEX input

C01E 8D F0      INHEX   BSR     INCH      GET A CHAR
C020 81 30              CMP A   #'0       ZERO
C022 2B 11              BMI     HEXERR    NOT HEX
C024 81 39              CMP A   #'9       NINE
C026 2F 0A              BLE     HEXRTS    GOOD HEX
C028 81 41              CMP A   #'A
C02A 2B 09              BMI     HEXERR    NOT HEX
C02C 81 46              CMP A   #'F
C02E 2E 05              BGT     HEXERR
C030 80 07              SUB A   #7        FIX A-F
C032 84 0F      HEXRTS  AND A   #$0F      CONVERT ASCII TO DIGIT
C034 39                 RTS

C035 7E C0 AF   HEXERR  JMP     CTRL      RETURN TO CONTROL LOOP
```

https://commons.wikimedia.org/wiki/File:Motorola_6800_Assembly_Language.png#/media/File:Motorola_6800_Assembly_Language.png

Paradigms a.k.a. the paths of programming

Imperative

- **procedural** (group into procedures, e.g. C) 🐍
- **object-oriented** (OOP, group into objects, e.g. C++) 🐍

Declarative

- **functional** (given by functions, e.g. Haskell) 🐍
- **logic** (rule system, e.g. Prolog)
- **mathematical** (mathematical optimization problem, e.g.)
- symbolic, ...

Imperative Python

- Step by step instructions

```
In_list = [1, 2]
out_list = []
for num in in_list:
    out_list = out_list + [num + 3]
print(out_list)
```

Procedural Python

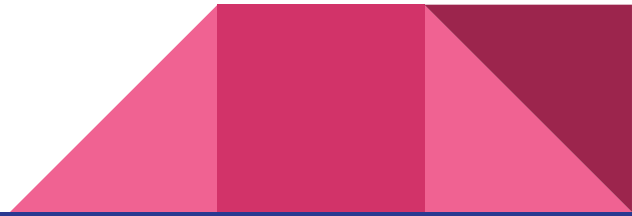
- Step by step instructions
- Wrapped into procedures
- Can cause (undesired) side-effects
  (see global variables)

```
def add_three(in_list):
    out_list = []
    for num in in_list:
        print(out_list)
        out_list = out_list + [num + 3]
    return(out_list)

print(add_three([1, 2]))
```

Functional Python

What was a functional example in Python?

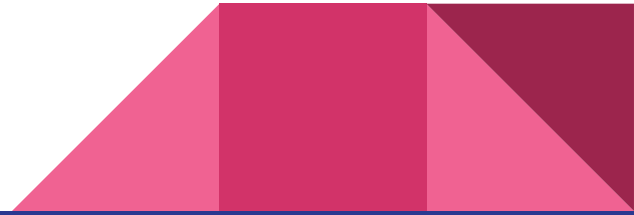


Functional Python

- Link a series of functions
- Lambda or general functions in Python and their chaining
- Helps to prevent side-effects as data are passed directly between functions
- Often immutable objects
- See functools package

```
print(list(map(lambda x: x + 3, [1, 2])))
```

What Python immutable object do you know?



Object-oriented Python

What classes did you already encounter in Python?

Object-oriented Python

- Separate procedures (methods) and data (attributes) into classes
- Allows for reuse (see inheritance)
- Initialize a member of the class (i.e. object)
- Private and public (given by dunders __ in Python)
- Special are __x__, “magic” methods
- Accessing and changing of private methods usually through “getter” and “setter” methods
- Useful in Python even just for own types (e.g. own style of dicts)

```
class Adder:
    def __init__(self, in_list):
        self._string = 'hello'
        self.__in_list = in_list
        self.out_list = []
    def add_three(self):
        for num in self.__in_list:
            self.out_list = self.out_list + [num + 3]

adder_object = Adder([1, 2])
adder_object.add_three()

print(adder_object.out_list)
print(adder_object._string)
print(adder_object.__in_list)
print(adder_object._Adder__in_list)
```

mangling

Exercise

- Create a general class Animal
  - With a private attribute `name` set at initialization
  - With a method `get_name` that returns the animals name
- Create a class that inherits from the Animal class (see guide)
- Let the Cat class have:
  - A private variable `purr_sound` that is a string for the sound of the cat's purring
  - A public method `purr` that prints out the `purr_sound`
- Show the initialization of a Cat object, print its name and make it purr.

Object-oriented principles

Encapsulation

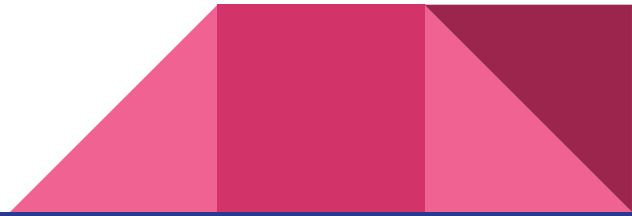
- Bundle data and methods that work on them, isolate them
- Objects as actors who “know” and “do” stuff
- Separate **what** x **how**
  What - what does the object do (i.e. *interface* of the object)
  How - actual implementation of the object

Polymorphism

- What an object does depends on the type or class of the object,
  i.e. we can call the same method but get different results. (Remember + ?)

Inheritance

- Hierarchy and reuse of classes and properties by the subclass (child) from the superclass (parent)
- Easier design and conceptualization of the problem



Symbolic Python

- Not native in Python(Sympy)
- Symbol manipulation based on an internal engine
  (i.e. not by the instructions of the programmer per se)
- For precise solutions (e.g., ⅓, π)
- Other symbolic tools are for example Mathematica or Maple

```
>>> sym.simplify((x + x * y) / x)
y + 1
```

Logical Python

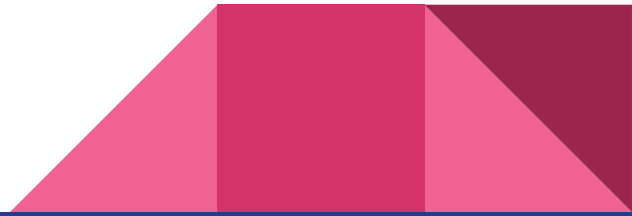
- Programmer provides only data
- The program = inference engine is fixed in advance
- Not native Python
  Sympy, Kanren

```
>>> from kanren import Relation, facts, run, var
>>> x = var()
>>> parent = Relation()
>>> facts(parent, ("Homer", "Bart"),
...               ("Homer", "Lisa"),
...               ("Abe",  "Homer"))

>>> run(1, x, parent(x, "Bart"))
('Homer',)

>>> run(2, x, parent("Homer", x))
('Lisa', 'Bart')
```

What other big distinction in programming languages do you remember?



Compiled

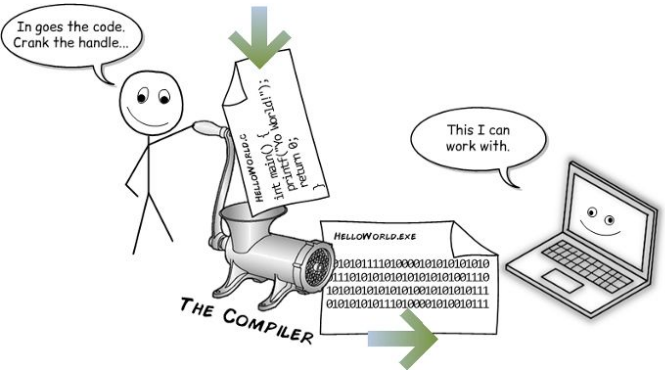
Interpreted

C, C++, Nim...

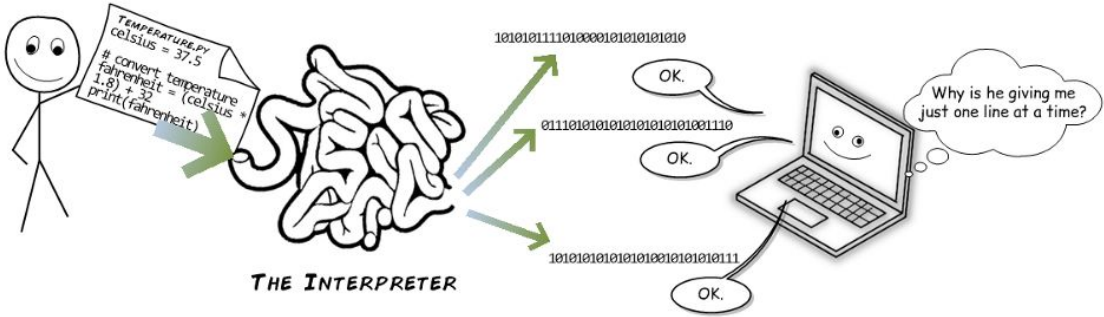
Java

Python

Clojure



vs



https://medium.com/young-coder/the-difference-between-compiled-and-interpreted-languages-d54f66aa71f0

Nim

- Syntax similar to Python
- Compiled language
- Many advanced features we did not cover in Python (e.g., references)

Note on references and side-effects in Python

See: https://blog.penjee.com/passing-by-value-vs-by-reference-java-graphical/

Note on references and side-effects in Python

```
def addInterest(balance, rate):
    balance = balance * (1 + rate)
    return balance

def test():
    amount = 1000
    rate = 0.05
    addInterest(amount, rate)
    print(amount)

test()
>>> 1000
```

vs

```
def addInterest(balances, rate):
    for i in range(len(balances)):
        balances[i] = balances[i] * (1+rate)

def test():
    amounts = [1000, 2200, 800, 360]
    rate = 0.05
    addInterest(amounts, 0.05)
    print(amounts)

test()

>>> [1050.0, 2310.0, 840.0, 378.0]
```

- Python passes values
- Value of mutable objects contains data that can be changed (e.g., lists)
- Can lead to undesired “side-effects”, i.e. unintended changes
- Other languages can have explicit reference or value passing

See: https://blog.penjee.com/passing-by-value-vs-by-reference-java-graphical/