

Programming in Python

Programming for Economists

Jeppe Druedahl

Learning phases

- **You have worked with:**
 1. DataCamp courses
 2. Lecture videos week 37-38
- **A lot of abstract new concepts!**
 1. OK to be confused now
 2. This is the last abstract stuff
 3. Afterwards it becomes more concrete and hands-on

Classes

- A **class** defines the **type** of an object
 - `.attribute`, state
 - `.method()`, action (incl. changing `self`)
- **Inheritance** (of methods) (`class Child(Parent)`)

```
1 class Parent:  
2     def __init__(self,value): self.value = value  
3     def double(self): return self.value * 2  
4  
5 class Child(Parent):  
6     def half(self): return self.value / 2  
7  
8 child = Child(10)  
9 print(child.value) # 10  
10 print(child.double()) # 20  
11 print(child.half()) # 5.0
```

References

- **Variables are references to an instance of an object**
- A **class** defines the **type** of an object
 - `.attribute`, state
 - `.method()`, action (incl. changing self)
- Inheritance (of methods) (class Child(Parent))
- = **assigns a reference** (*not a copy!*)

Question: What does a end up as? What if a = [1,2,3]?

```
1 a = np.array([1,2,3])
2 b = a
3 c = a[1:] # slicing
4 b[0] = 3 # indexing
5 c[0] = 3
```

Types and in-place operations

- **Atomic types:** int, float, str, bool, etc.
- **Containers** list, tuple, dict, set, np.array, etc.
- **Mutables** (e.g. list, np.array) can change in-place
 1. **In-place operators** ($+=$, $-=$ etc.)
 2. **Slicing**: $x[:] = x + y$
- **Immutables** (e.g. atomic types and tuples) can never change

Questions: What does y end up as?

```
1 x = np.array([1,2,3])
2 y = x
3 x += 1
4 x[:] = x + 1
5 x = x + 1
```

Functions and scope

- **Functions are objects** (can e.g. be arguments in functions)
Unlike in math:
 1. Can change its arguments (side-effects)
 2. Can call itself (recursion)
- Variables can both be **local scope** (good) or **global scope** (bad)

Questions: What is the output?

```
1 a = 1
2 def f(x):
3     return x+a
4 print(f(1))
5 a = 2
6 print(f(1))
```

Computational tree and branches

- **Comparison** ($==$, \neq , $<$, \leq , `not`, `and`, `or` etc.)
- **Conditionals** (`if`, `elif`, `else`)
- **Loops** (`for`, `while`, `continue`, `break`)
- **Convergence** (tolerance in optimizer or root-finder/equation-solver)

Questions: How could this be implemented with a while loop?

```
1 x = x0
2 for i in range(n):
3     y = evaluate(x)
4     if check(y): break
5     x = update(x,y)
6 else:
7     raise ValueError('did not converge')
```

Decimal numbers are not exact

- Never use exactness for decimal numbers
 - Order of computation matter
 - Best with numbers are around 1 (underflow and overflow)
- Division, exp, log etc. are (costly) approximations
- **Function approximation and interpolation often needed**

Questions: Which are True and which are False?

```
1 print(0.1 + 0.2 == 0.3)
2 print(0.5 + 0.5 == 1.0)
3 print(np.isclose(0.1+0.2,0.3))
4 print(np.isclose(1e-200*1e200*1e200*1e-200,1.0))
5 print(np.isinf(1e-200*(1e200*1e200)*1e-200))
6 print(np.isclose(1e200*(1e-200*1e-200)*1e200,0.0))
```