PyQak - Michele Proverbio

Syntax example

```
import pyqak
from pyqak import *
from transitions import *
from context import Context, ExternalContext

ctx_radar = ExternalContext('localhost', 8020)
ctx_radar.external_actor('radar')

ctx = Context('localhost', 8030)
ctx.actor_scope('robot')

@initial
@state
async def init(self, t):
    print('robot | init')
    await self.transition('work', Epsilon)

@state
async def work(self, t):
    print('robot | work')
    print('w')  # fire motors
    await self.transition('halt', WhenEvent, 'sonar', lambda s: int(s) < 10)
    await self.transition('toradar', WhenEvent, 'sonar', lambda s: int(s) >= 10)

@state
async def halt(self, t):
    print('robot | halt')
    print('h')  # stop motors

@state
async def toradar(self, t):
    print('robot | toradar')
    await self.request('radar', 'polar', t['msg'].payload)
    await self.transition('halt', WhenEvent, 'sonar', lambda s: int(s) < 10)
    await self.transition('handle_response', WhenReply, 'polarReply')

@state
async def handle_response(self, t):
    print('robot | handle_response')
    await self.transition('work', Epsilon)

pyqak.run()
```

Kotlin coroutines vs Python Asyncio

launching jobs

Kotlin

explicit context.

```
fun launcher(){
    GlobalScope.launch {coroutine()}
}
```

implicit context.

ie. inside a suspend function declaration or a runBlocking wrapper.

```
fun launcher() = runBlocking {
    launch {coroutine()}
}
```

Python

explicit context.

```
def launcher():
    asyncio.launch (coroutine())
```

implicit context.

ie. inside an async function declaration

```
async def launcher():
    await coroutine()
```

suspendable routines

Kotlin

```
suspend fun ioBoundFun(){
    delay(1000)
    println("IO operation | Done")
}
```

Python

```
async def ioBoundFun():
    await sleep(1)
    print("IO operation | Done")
```

async jobs/tasks and promises/futures

Kotlin

GlobalScope.async() is a builder of jobs. It returns a promise that can be used to "join" the async execution.

```
suspend fun activate(){
    val job1 = GlobalScope.async {
        ioBoundFun()
    }
    val job2 = GlobalScope.async {
        ioBoundFun()
    }
    if(! job1.isCompleted || ! job2.isCompleted)
        println("Waiting for completion")
    val end1 = job1.await()
    val end2 = job2.await()
    println("All jobs done")
}
```

Python

asyncio.create_task() is the counter part of GlobalScope.async() with the same properties.

```
async def activate():
    task0 = asyncio.create_task (ioBoundFun())
    task1 = asyncio.create_task (ioBoundFun())
    await task0
    await task1
```

By Michele Proverbio email: michele.proverbio@studio.unibo.it