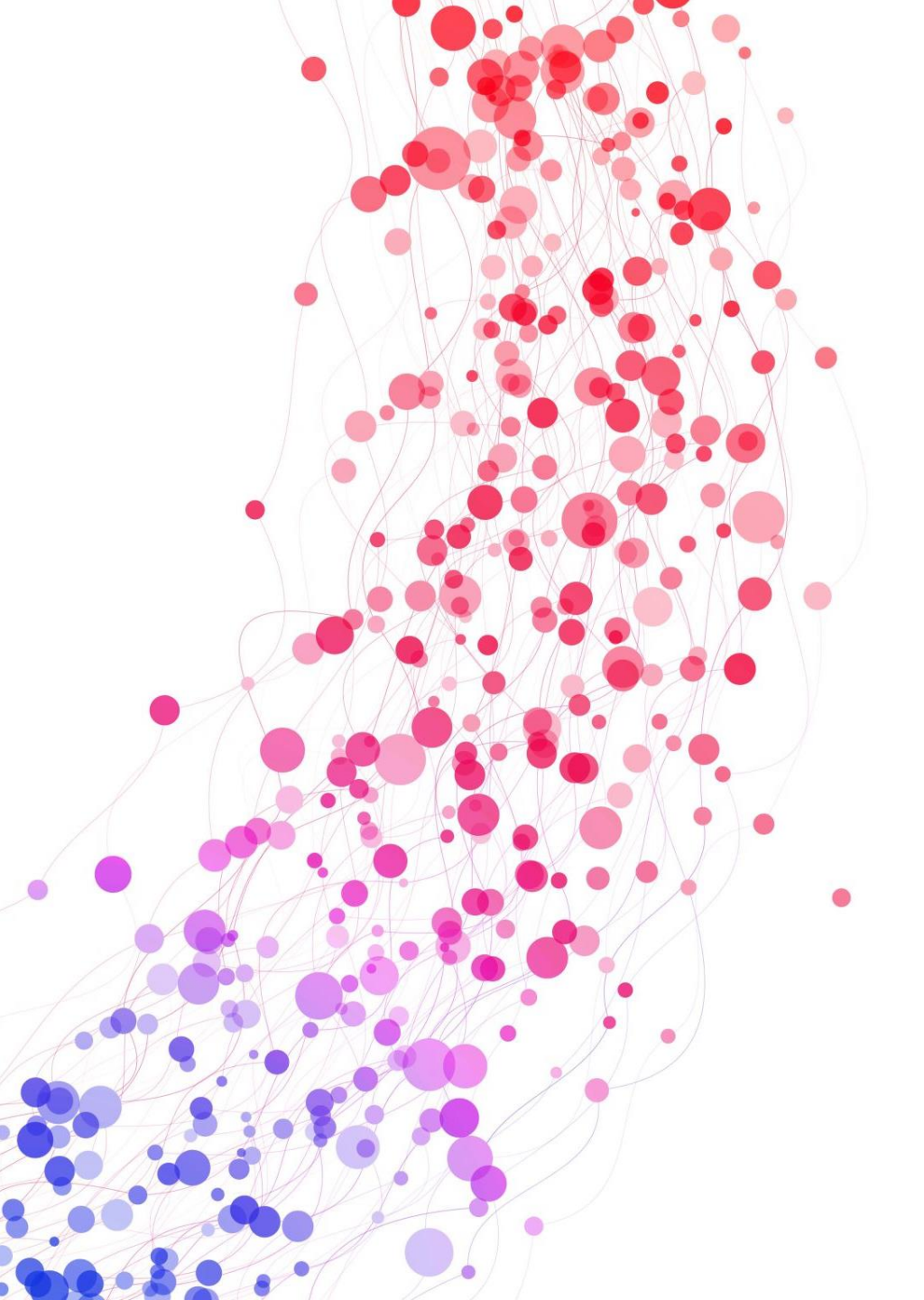


PROGRAMMING PARADIGMS IN PYTHON

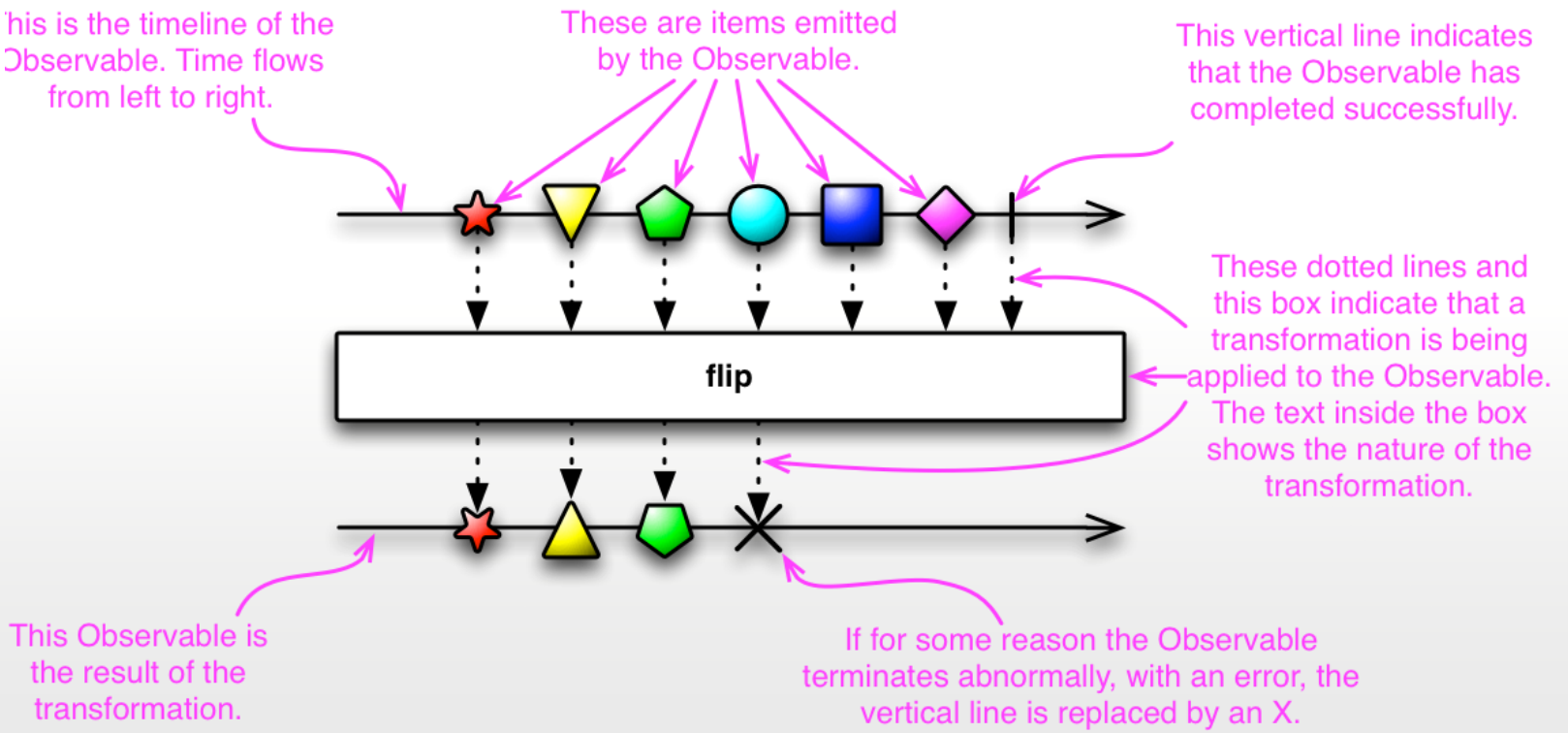
(FUNCTIONAL & REACTIVE)

MUTHUKUMARAN NAVANEETHAKRISHNAN

SESSION III

- Observing and printing
- Using Schedulers with Create
- Creating Observables
  - from
  - Just
- Subjects
  - Subject
  - ReplaySubject
  - BehaviorSubject
- Transforming Observables

- Transforming Observables
- Aggregate Operators
- Error Handling
- Combining Observable
- Testing Observables


'his is the timeline of the
Observable. Time flows
from left to right.
These are items emitted
by the Observable.
This vertical line indicates
that the Observable has
completed successfully.
flip
These dotted lines and
this box indicate that a
transformation is being
applied to the Observable.
The text inside the box
shows the nature of the
transformation.
This Observable is
the result of the
transformation.
If for some reason the Observable
terminates abnormally, with an error, the
vertical line is replaced by an X.

OBSERVING AND PRINTING

- Manually Defining
  - on_next
  - on_error
  - on_completed
- Via Class with methods
  - on_next
  - on_error
  - on_completed

OBSERVING AND PRINTING DEMO

- Use Rx.create with two sources
  - All male users
  - All female user
  - Print data separately based on name of subscription

SCHEDULERS IN RX CREATE

- Schedulers define the way to run the source
  - In Different Thread
- Specify the Scheduler on subscribe as a separate parameter
- ThreadPoolScheduler
  - Used to do a worker model , which will be started by default

SCHEDULERS IN RX CREATE DEMO

- Use Rx.create with source
  - emit the items with & without scheduler

CREATING WITH JUST & FROM

- just – Invoke all the data at a time and fire completed
- from_list – Invoke list of items one at a time and fire completed

JUST & FROM DEMO

- Load products from product.json
- Using just & from_list
  - Emit List of users
  - Observe How data is printed for each observable

SUBJECT

- Most of you are aware
  - An *observer subscribes* to an *Observable*.
  - observer reacts to whatever item or sequence of items the Observable *emits*.
- What if you need to be an observer & Observable
  - A way to put data whenever you need
  - A way to subscribe those data changes as well?

SUBJECT VARIANCES

- Subject – Latest value is emitted to all the subscribers
- ReplaySubject – all the values based on the size will be emitted to all subscribers
- BehaviourSubject – Will hold a default value and latest value is emitted to all the subscribers
- AsyncSubject – Only the last value will be emitted , after the on_complete method invoked

SUBJECT DEMO

- Demonstrate List of users with
  - Subject -
  - ReplaySubject
  - BehaviourSubject
  - AsyncSubject

MAP

- Converting one form to other

MAP DEMO

```
python_dev_salary = 1000000
python_dev_salary_in_bangalore = python_dev_salary + (python_dev_salary * (26/100))
python_dev_salary = 90000
```

MAP DEMO

```
python_dev_salary = 1000000
python_dev_salary_in_bangalore = python_dev_salary + (python_dev_salary * (26/100))
python_dev_salary = 90000
```

```
python_dev_salary = BehaviorSubject(1000000)
python_dev_salary_in_bangalore = python_dev_salary.pipe(
    ops.map(increment_by_26_percent)
)
python_dev_salary.on_next(90000)
```

REACTIVE PROGRAMMING DEMO

```
python_dev_salary = BehaviorSubject(1000000)
```

Subject

```
 python_dev_salary_in_bangalore = python_dev_salary.pipe(
  ops.map(increment_by_26_percent)
)
```

Observable

```
python_dev_salary.subscribe(print_python_dev_salary)
python_dev_salary_in_bangalore.subscribe(print_ python_dev_salary_in_bangalore)
```

Observer

```
python_dev_salary.on_next(90000)
```

Push data to Subject

TRANSFORMATION

- filter - filter a set of data
- take - Take only first x elements
- distinct – show only distinct fields from a stream of data
- flatmap – flatten array of array to array

TRANSFORMATION DEMO

- filter - filter only products of Footwear category
- take - Take first 3 elements of footwear category
- flatmap – List Down all the users within review as one single list
- distinct – Identify unique list of users within review

AGGREGATION

- reduce – reduce a group of elements to one
- count – count of elements
- max – maximum of elements
- min – minimum of elements
- average – average of elements
- sum – sum of elements

AGGREGATION DEMO

- reduce – find total price of footwear category
- count – find count of reviews for Watches category
- max – find Highest price in Clothing category
- min – find which product has Lowest dealPrice

ERROR HANDLING

- retry – retry the same observable again
- catch – catch error within observable

ERROR HANDLING DEMO

- Connect an api to retrieve product buy url for a product
- Retry that api , when the api is down for 3 times
  - Retry
- Provide a default response if the api is down after 3 times
  - catch

COMBINING OBSERVABLES

- merge – emit only the changed value
- combineLatest – if one of the observables is updated or new, emit it along with all other observable value
- zip – emit only if all the items are updated and emit as pair

COMBINING OBSERVABLES DEMO

- Preferences
  - Language
  - Theme
- Update Only the chosen Preference
- Update All the Preferences , even if one preference is changed
- Update Only all the preferences are changed

TESTING OBSERVABLE

- Observable.run() => Produce Blocking result , Usually the last emitted result
- `to_iterable()`
  - `Part of operators`
  - `Connect through pipe to get all the results from start to end`

TESTING OBSERVABLE DEMO

- Write test cases for the observables created in Just & From Section

TESTING SUBJECTS

- TestScheduler => Schedule the subjects to be observed with MockObserver
- `Assert the values received by mockObserver`

TESTING SUBJECT DEMO

- Write test cases for the subjects created in map section