PrettyPrinting

Optimal layout for code and data

https://github.com/MechanicalRabbit/PrettyPrinting.jl

Kyrylo Simonov

JuliaCon 2021

```
julia> print(["Hello", "World"])
["Hello", "World"]
```

```
julia> print(["Hello", "World"])
["Hello", "World"]
```

```
julia> show(["Hello", "World"])
["Hello", "World"]
```

```
julia> print(["Hello", "World"])
["Hello", "World"]
```

```
julia> show(["Hello", "World"])
["Hello", "World"]
```

```
julia> display(["Hello", "World"])
2-element Vector{String}:
 "Hello"
 "World"
```

```
julia> print(["Hello", "World"])
["Hello", "World"]
```

```
julia> show(["Hello", "World"])
["Hello", "World"]
```

```
julia> display(["Hello", "World"])
2-element Vector{String}:
 "Hello"
 "World"
```

```
julia> dump(["Hello", "World"])
Array{String}((2,))
  1: String "Hello"
  2: String "World"
```

```
julia> print(["Hello", "World"])
["Hello", "World"]
```

```
julia> show(["Hello", "World"])
["Hello", "World"]
```

```
julia> using PrettyPrinting

julia> pprint(["Hello", "World"])
["Hello", "World"]
```

```
julia> display(["Hello", "World"])
2-element Vector{String}:
 "Hello"
 "World"
```

```
julia> dump(["Hello", "World"])
Array{String}((2,))
  1: String "Hello"
  2: String "World"
```

```
$ cat Project.toml
name = "FunSQL"
uuid = "cf6cc811-59f4-4a10-b258-a8547a8f6407"
authors = ["Kirill Simonov <xi@resolvent.net>", "Clark C. Evans <cce@clarkevans.com>"]
version = "0.6.0"

[deps]
Dates = "ade2ca70-3891-5945-98fb-dc099432e06a"
PrettyPrinting = "54e16d92-306c-5ea0-a30b-337be88ac337"

[compat]
julia = "1.4"
PrettyPrinting = "0.3.2"
```

```
julia> TOML.parsefile("Project.toml") |> show
Dict{String, Any}("deps" => Dict{String, Any}("Dates" => "ade2ca7
0-3891-5945-98fb-dc099432e06a", "PrettyPrinting" => "54e16d92-306
c-5ea0-a30b-337be88ac337"), "name" => "FunSQL", "uuid" => "cf6cc8
11-59f4-4a10-b258-a8547a8f6407", "compat" => Dict{String, Any}("j
ulia" => "1.4", "PrettyPrinting" => "0.3.2"), "authors" => ["Kiri
ll Simonov <xi@resolvent.net>", "Clark C. Evans <cce@clarkevans.c
om>"], "version" => "0.6.0")
```

```
julia> TOML.parsefile("Project.toml") |> show
Dict{String, Any}("deps" => Dict{String, Any}("Dates" => "ade2ca7
0-3891-5945-98fb-dc099432e06a", "PrettyPrinting" => "54e16d92-306
c-5ea0-a30b-337be88ac337"), "name" => "FunSQL", "uuid" => "cf6cc8
11-59f4-4a10-b258-a8547a8f6407", "compat" => Dict{String, Any}("j
ulia" => "1.4", "PrettyPrinting" => "0.3.2"), "authors" => ["Kiri
ll Simonov <xi@resolvent.net>", "Clark C. Evans <cce@clarkevans.c
om>"], "version" => "0.6.0")
```

```
julia> TOML.parsefile("Project.toml") |> display
Dict{String, Any} with 6 entries:
  "deps"    => Dict{String, Any}("Dates"=>"ade2ca70-3891-5945-98…
  "name"    => "FunSQL"
  "uuid"    => "cf6cc811-59f4-4a10-b258-a8547a8f6407"
  "compat"  => Dict{String, Any}("julia"=>"1.4", "PrettyPrinting…
  "authors" => ["Kirill Simonov <xi@resolvent.net>", "Clark C. E…
  "version" => "0.6.0"
```

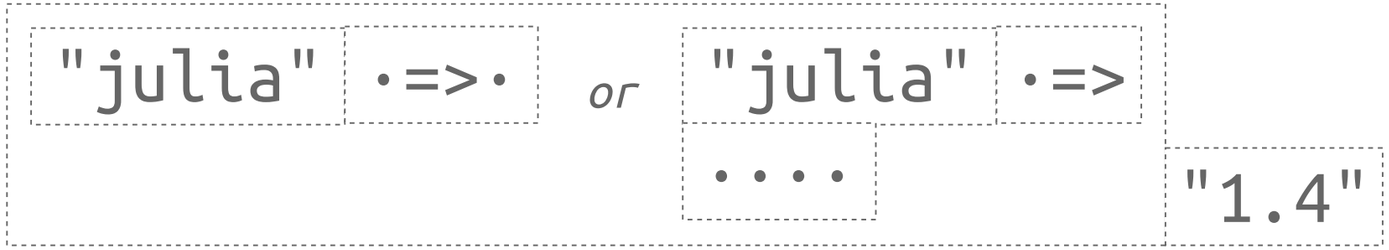
```
julia> TOML.parsefile("Project.toml") |> show
Dict{String, Any}("deps" => Dict{String, Any}("Dates" => "ade2ca7
0-3891-5945-98fb-dc099432e06a", "PrettyPrinting" => "54e16d92-306
```

```
julia> TOML.parsefile("Project.toml") |> pprint
Dict("deps" => Dict("Dates" =>
                    "ade2ca70-3891-5945-98fb-dc099432e06a",
                "PrettyPrinting" =>
                    "54e16d92-306c-5ea0-a30b-337be88ac337"),
     "name" => "FunSQL",
     "uuid" => "cf6cc811-59f4-4a10-b258-a8547a8f6407",
     "compat" => Dict("julia" => "1.4",
                      "PrettyPrinting" => "0.3.2"),
     "authors" => ["Kirill Simonov <xi@resolvent.net>",
                   "Clark C. Evans <cce@clarkevans.com>"],
     "version" => "0.6.0")
```

```
julia> TOML.parsefile("Project.toml") |> pprint
Dict("deps" => Dict("Dates" =>
                        "ade2ca70-3891-5945-98fb-dc099432e06a",
                    "PrettyPrinting" =>
                        "54e16d92-306c-5ea0-a30b-337be88ac337"),
     "name" => "FunSQL",
     "uuid" => "cf6cc811-59f4-4a10-b258-a8547a8f6407",
     "compat" => Dict("julia" => "1.4",
                      "PrettyPrinting" => "0.3.2"),
```

```
julia> TOML.parsefile("Project.toml") |> pprint
Dict("deps" => Dict("Dates" => "ade2ca70-3891-5945-98fb-dc099432e06a",
                    "PrettyPrinting" => "54e16d92-306c-5ea0-a30b-337be88ac337"),
     "name" => "FunSQL",
     "uuid" => "cf6cc811-59f4-4a10-b258-a8547a8f6407",
     "compat" => Dict("julia" => "1.4", "PrettyPrinting" => "0.3.2"),
     "authors" => ["Kirill Simonov <xi@resolvent.net>",
                   "Clark C. Evans <cce@clarkevans.com>"],
     "version" => "0.6.0")
```


```
julia> PrettyPrinting.tile("julia" => "1.4")
(literal("\"julia\"") * literal(" => ") |
 literal("\"julia\"") * literal(" =>") / indent(4)) *
literal("\"1.4\"")
```



Phillip Yelland, A New Approach to Optimal Code Formatting, 2016

Supported data types:

- `Tuple`
- `NamedTuple`
- `Vector`
- `Set`
- `Dict`
- `Expr`

Supported data types:

- `Tuple`
- `NamedTuple`
- `Vector`
- `Set`
- `Dict`
- `Expr`

```
julia> Pkg.project()
Pkg.API.ProjectInfo("FunSQL", UUID("cf6cc811-59f4-4a10-b258-a8547
a8f6407"), v"0.6.0", true, Dict{String, Base.UUID}("Dates" => UUI
D("ade2ca70-3891-5945-98fb-dc099432e06a"), "PrettyPrinting" => UU
ID("54e16d92-306c-5ea0-a30b-337be88ac337")), "/home/xi/Mechanical
Rabbit/FunSQL.jl/Project.toml")
```

```
julia> Pkg.project()
Pkg.API.ProjectInfo("FunSQL", UUID("cf6cc811-59f4-4a10-b258-a8547
a8f6407"), v"0.6.0", true, Dict{String, Base.UUID}("Dates" => UUI
D("ade2ca70-3891-5945-98fb-dc099432e06a"), "PrettyPrinting" => UU
ID("54e16d92-306c-5ea0-a30b-337be88ac337")), "/home/xi/Mechanical
Rabbit/FunSQL.jl/Project.toml")
```

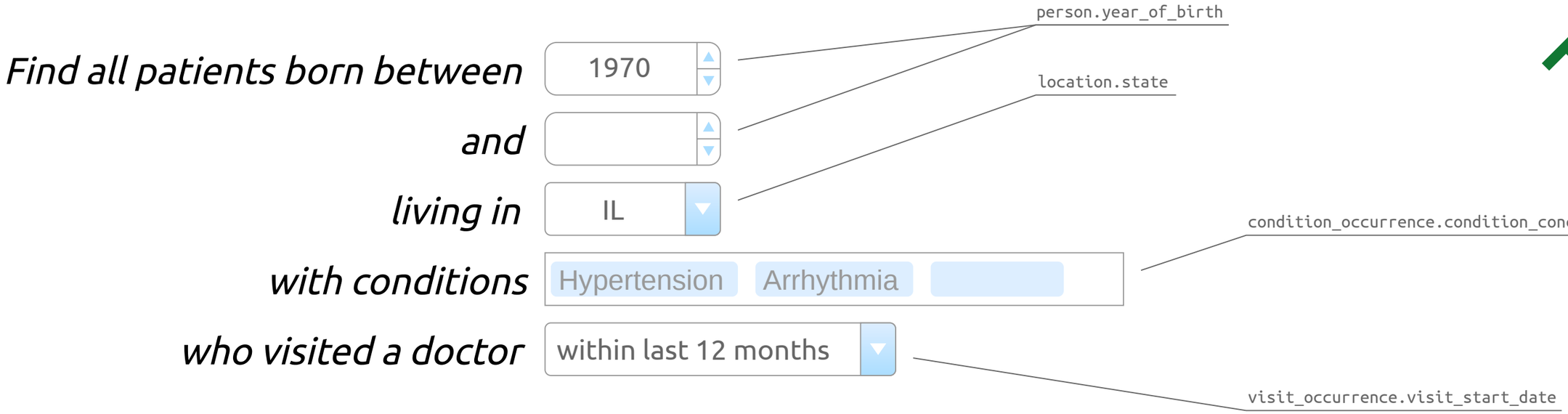
```
PrettyPrinting.quoteof(info::Pkg.API.ProjectInfo) =
    :(Pkg.API.ProjectInfo(name = $(info.name),
                          uuid = $(info.uuid),
                          version = $(info.version),
                          ispackage = $(info.ispackage),
                          dependencies = $(info.dependencies),
                          path = $(info.path)))
```

```
PrettyPrinting.quoteof(info::Pkg.API.ProjectInfo) =
    :(Pkg.API.ProjectInfo(name = $(info.name),
                          uuid = $(info.uuid),
                          version = $(info.version),
                          ispackage = $(info.ispackage),
                          dependencies = $(info.dependencies),
                          path = $(info.path)))
```

```
julia> Pkg.project() |> pprint
Pkg.API.ProjectInfo(
    name = "FunSQL",
    uuid = UUID("cf6cc811-59f4-4a10-b258-a8547a8f6407"),
    version = v"0.6.0",
    ispackage = true,
    dependencies =
        Dict("Dates" =>
                 UUID("ade2ca70-3891-5945-98fb-dc099432e06a"),
             "PrettyPrinting" =>
                 UUID("54e16d92-306c-5ea0-a30b-337be88ac337")),
    path = "/home/xi/MechanicalRabbit/FunSQL.jl/Project.toml")
```

```
Base.show(io::IO, ::MIME"text/plain", info::Pkg.API.ProjectInfo) =
    pprint(io, info)
```

```
julia> Pkg.project()
Pkg.API.ProjectInfo(
    name = "FunSQL",
    uuid = UUID("cf6cc811-59f4-4a10-b258-a8547a8f6407"),
    version = v"0.6.0",
    ispackage = true,
    dependencies =
        Dict("Dates" =>
                 UUID("ade2ca70-3891-5945-98fb-dc099432e06a"),
             "PrettyPrinting" =>
                 UUID("54e16d92-306c-5ea0-a30b-337be88ac337")),
    path = "/home/xi/MechanicalRabbit/FunSQL.jl/Project.toml")
```




```
FindPatients(; start_year = nothing, end_year = nothing,
               state = nothing,
               condition_concepts = [],
               latest_visit_threshold = nothing) =
    From(person) |>
    FilterByYearOfBirth(; start_year, end_year) |>
    FilterByState(; state) |>
    FilterByConditions(; condition_concepts) |>
    FilterByLatestVisit(; latest_visit_threshold) |>
    Select(Get.person_id)
```

FunSQL

```
julia> FindPatients(start_year = 1970, state = "IL") |> render
ERROR: GetError: cannot find locatin_id in:
let person = SQLTable(:person, …),
    location = SQLTable(:location, …),
    q1 = From(person),
    q2 = q1 |> Where(Fun.">="(Get.year_of_birth, Lit(1970))),
    q3 = From(location),
    q4 = q3 |> Where(Fun."=="(Get.state, Lit("IL"))),
    q5 = q2 |>
         Join(q4 |> As(:location),
              Fun."=="(Get.location_id,
                       Get.location.locatin_id)),
    q6 = q5 |> Select(Get.person_id)
    q6
end
```