PDL Quick Reference

```
pip install prompt-declaration-language
```

```
pdl examples/hello/hello.pdl
```

LLM call with current context

```yaml
model: watsonx/ibm/granite-13b-chat-v2
parameters:
  temperature: 0.1
```

LLM call with explicit input

```yaml
model: watsonx/ibm/granite-13b-chat-v2
parameters:
  temperature: 0.1
input:
  array:
    - role: user
      content: Hello,
```

Reading from a file or stdin

```yaml
read:  # optionally, add file name
message: Please enter an input.
multiline: true  # omit to stop at \n
```

Creating data (v1, v2 can be any block)

```yaml
text:  # outputs "v1v2"
  - v1
  - v2
```

```yaml
lastOf:  # outputs v2
  - v1
  - v2
```

```yaml
array:  # outputs [v1, v2]
  - v1
  - v2
```

```yaml
object:  # outputs {k1: v1, k2: v2}
  k1: v1
  k2: v2
```

```yaml
data:  # outputs {k1: v1, model: v2}
  k1: v1
  model: v2  # no LLM call
```

Including a PDL file

```yaml
include: ./helper_defs.pdl
```

Declaring and calling functions

```yaml
def: add
function:
  x: int
  y: int
return: ${x + y}
```

```yaml
call: ${add}
args:
  x: 2
  y: 2
  pdl_context: []  # optional
```

Control constructs

```yaml
if: ${x > 0}
then: positive
else: non-negative
```

```yaml
match: ${x}
with:
  - case: one
    then: 1
  - case: two
    then: 2
```

```yaml
for:  # outputs 2_0_5
  i: [1, 0, 1]
  j: [2, 3, 5]
repeat: ${i * j}
join:
  with: _  # optional
```

```yaml
repeat:  # outputs HiHiHi
  text: Hi
num_iterations: 3
```

```yaml
repeat:
  def: x
  read:
until: ${(x | trim) == "stop"}
```

Optional keywords for any block

```yaml
description: documentation text
```

```yaml
def: x  # define variable from block
```

```yaml
defs:  # define multiple variables
  x: v1
  y: v2
```

```yaml
role: user  # or system or assistant
```

```yaml
contribute: [result, context]  # or less
```

```yaml
parser: json  # or jsonl, yaml, regex
```

```yaml
spec: type  # type specification
```

Executing code

```yaml
lang: python  # or jinja, pdl
code: result = "Hello, world!"
```

spec Types (shorthand for JSON Schema)

Basic types	`str, int, float, bool, null`
Arrays	`[int]`
Objects	`{x: int, y: int}`
Enums	`{enum: [red, green, blue]}`

${…} Expressions (subset of Jinja2)

Basic values	`"hi", 5, 3.1, true, none`
Arrays	`[1, 2, 3]`
Objects	`{"x": 4, "y": 5}`
Variables	`x, y[0], z.f`
Operators	`+, -, *, /, //, %, **, ~, and, or, not, ==, <, >, in`
Tests	`x if x is defined else 0`
Filters	`x | default(0)`