Set Design Document

New expression

Grammar

```
literal :=
...
| lambda [<name> [, <name>]*]? : <expr>
```

AST

```
// To stay a subset of Python, lambda parameters cannot have type hints,
// so we defer the type-checking of an anonymous function to the point of
use
// necessitating this new `LambdaParam` type
type LambdaParam = { name: string, type?: Type }
type Literal<A> =
...
| { a?: A, tag: "lambda", params: Array<LambdaParam>, expr: Expr<A> }
```

New Type **(parsing requires "@lezer/python": "^0.16.0", or elbow grease)**

Grammar

```
type :=
...
| Callable[[ [<type> [, <type>]* ]? ], <type> ]
```

AST

```
type Type =
...
| { tag: "callable", params: Array<Type>, ret: Type }
```

Test Cases

1. Z-combinator

```
make_rec: Callable[
    [
      Callable[
        [Callable[[int], int]],
        Callable[[int], int]
      ]
    ],
    Callable[[int], int]
  ] = (lambda g:
  ( lambda rec: g(lambda y: rec(rec)(y)) )
  ( lambda rec: g(lambda y: rec(rec)(y)) ))
fact: Callable[[int], int] = make_rec(lambda rec: lambda x:
  1 if x == 0 else rec(x - 1) * x)
print(fact(5))
```

Should pass, yielding `120`.