LAMBDA EXPRESSIONS

What are Lambda Expressions?

We already know how to create functions using the `def` statement

Lambda expressions are simply another way to create functions anonymous functions

keyword

parameter list optional

the `:` is required, even for zero arguments

```
lambda [parameter list]: expression
```

this expression is evaluated and returned when the lambda function is called

(think of it as the "body" of the function)

the expression returns a function object

that evaluates and returns the `expression` when it is called

it can be assigned to a variable

passed as an argument to another function

it is a function, just like one created with `def`

Examples

```
lambda x: x**2

lambda x, y: x + y

lambda : 'hello'

lambda s: s[::-1].upper()
```

`type(lambda x: x**2)` → `function`

Note that these expressions are function objects, but are not "named"

→ anonymous functions

Lambdas, or anonymous functions, are NOT equivalent to closures

Assigning a Lambda to a Variable Name

```
my_func = lambda x: x**2

type(my_func)   → function

my_func(3)      → 9

my_func(4)      → 16
```

identical to:

```
def my_func(x):
    return x**2

type(my_func)   → function

my_func(3)      → 9

my_func(4)      → 16
```

Passing as an Argument to another Function

```
def apply_func(x, fn):
    return fn(x)



apply_func(3, lambda x: x**2)              → 9

apply_func(2, lambda x: x + 5)             → 7

apply_func('abc', lambda x: x[1:] * 3)     → bcbcbc
```

equivalently:

```
def fn_1(x):
    return x[1:] * 3

apply_func('abc', fn_1)    → bcbcbc
```

Limitations

The "body" of a `lambda` is limited to a single expression

no assignments

```
lambda x: x = 5
```
✗

```
lambda x: x = x + 5
```
✗

no annotations

```
def func(x: int):
    return x**2
```
✓

```
lambda x:int : x*2
```
✗

single logical line of code → line-continuation is OK, but still just one expression

```
lambda x: x * \
    math.sin(x)
```
✓

Code