The try / except Structure

THE TRY / EXCEPT STRUCTURE

- You surround a dangerous section of code with try and except
- If the code in the try works - the except is skipped
- If the code in the try fails - it jumps to the except section

THE TRY / EXCEPT STRUCTURE

- What will happen if you run the following code?

```
astr = 'Hello Bob'
istr = int(astr)
print('First', istr)
astr = '123'
istr = int(astr)
print('Second', istr)
```

THE TRY / EXCEPT STRUCTURE

- What will happen if you run the following code?

```
astr = 'Hello Bob'
istr = int(astr)
print('First', istr)
astr = '123'
istr = int(astr)
print('Second', istr)
```

- You will run into an error message (red first two lines) and the rest of the code will not run.

THE TRY / EXCEPT STRUCTURE

```
astr = 'Hello Bob'
try:
    istr = int(astr)
except:
    istr = -1

print('First', istr)

astr = '123'
try:
    istr = int(astr)
except:
    istr = -1

print('Second', istr)
```

When the first conversion fails - it just drops into the except: clause and the program continues.

When the second conversion succeeds - it just skips the except: clause and the program continues.

BUILT IN ERROR

- just run:
  print(x)

- You will receive error message

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-81745ac23551> in <module>()
----> 1 print(x)

NameError: name 'x' is not defined
```

- Error type: NameError

SMOOTH CODING

this way, you run the code smoothly

```
try:
    print(x)
except NameError:
    print('x is not assigned')
    error = -1
if(error == -1):
    x = 100
    print(x)
```

ERROR TYPES

- You can find all of them here.
  https://docs.python.org/3/tutorial/errors.html

- eg:
  When writing robust code, say load some data.
  You can use try/except to preprocess the data

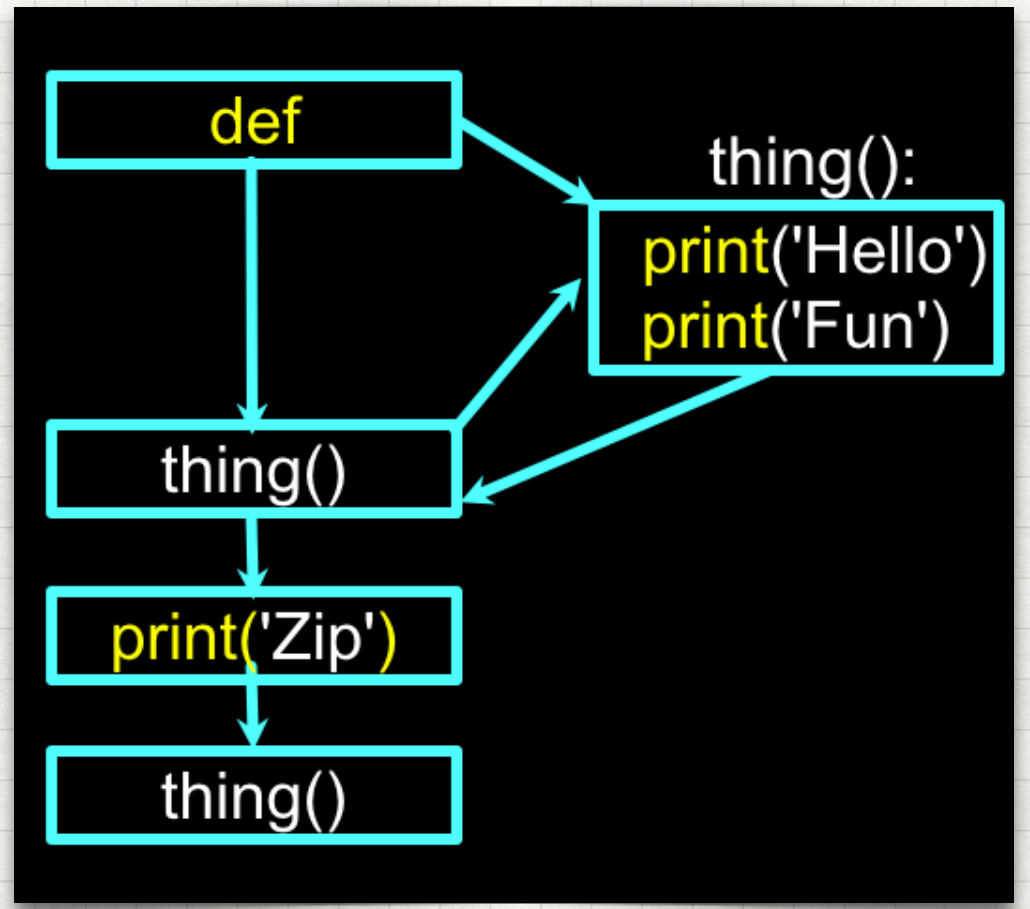
FUNCTIONS

STORED (AND REUSED) STEPS

- Program:

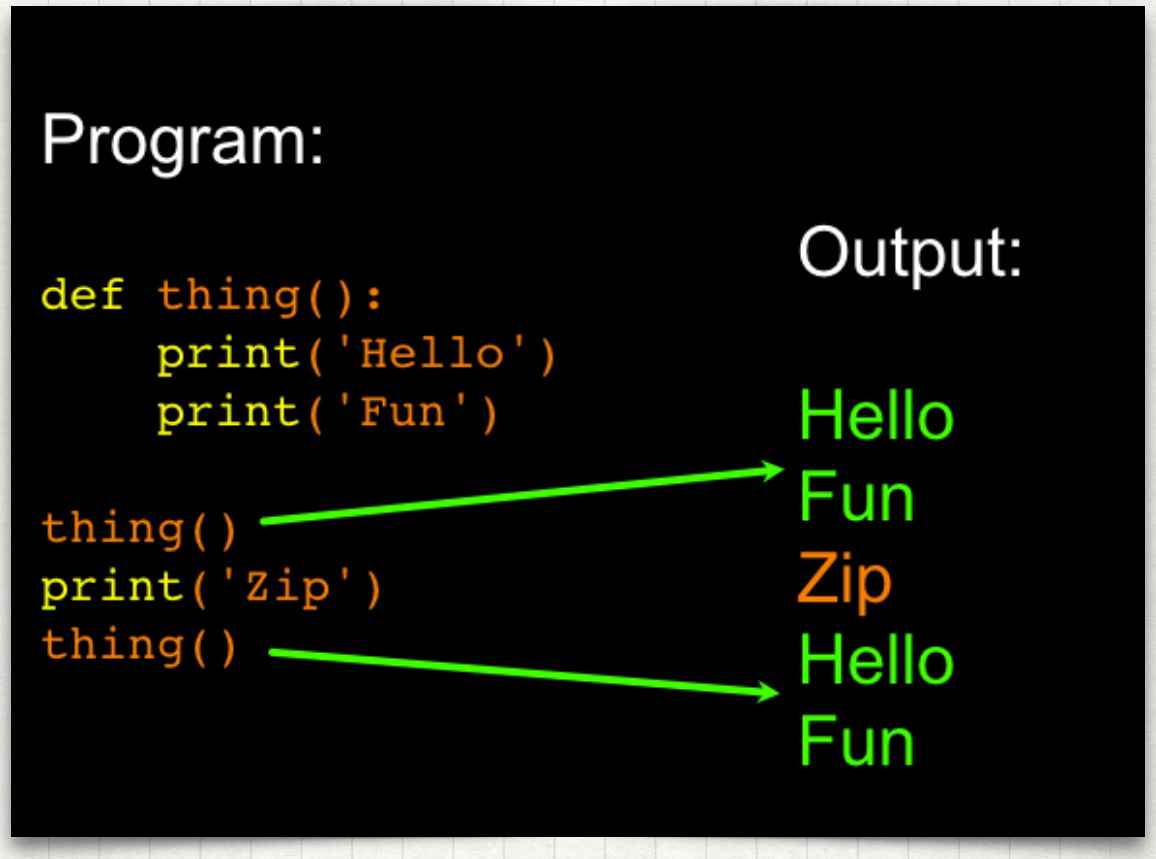
```
def thing():
    print('Hello')
    print('Fun')

thing()
print('Zip')
thing()
```



STORED (AND REUSED) STEPS

- function without input



PYTHON FUNCTIONS

- There are two kinds of functions in Python.
  - - Built-in functions that are provided as part of Python - print(), input(), type(), float(), int() ...
  - - Functions that we define ourselves and then use
- We treat the built-in function names as “new” reserved words (i.e., we avoid them as variable names)

FUNCTION DEFINITION

- In Python a function is some reusable code that takes arguments(s) as input, does some computation, and then returns a result or results
  function: print
  takes argument: (123)
  returns: 123

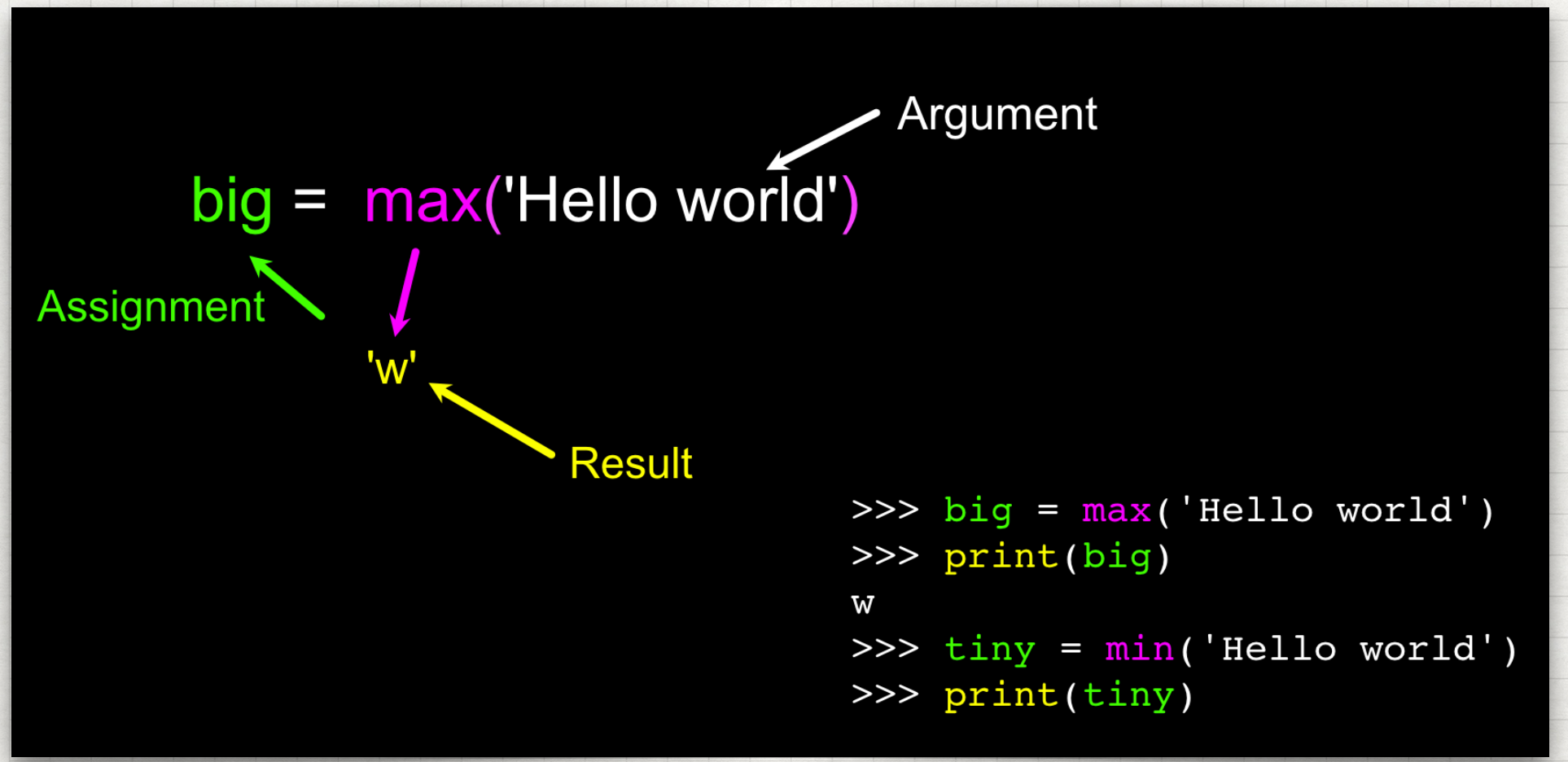
- We define a function using the def reserved word:
  thing()

- We call/invoke the function by using the function name, parentheses, and arguments in an expression

BUILT-IN FUNCTIONS

- You can find them here: https://docs.python.org/3/library/functions.html

		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

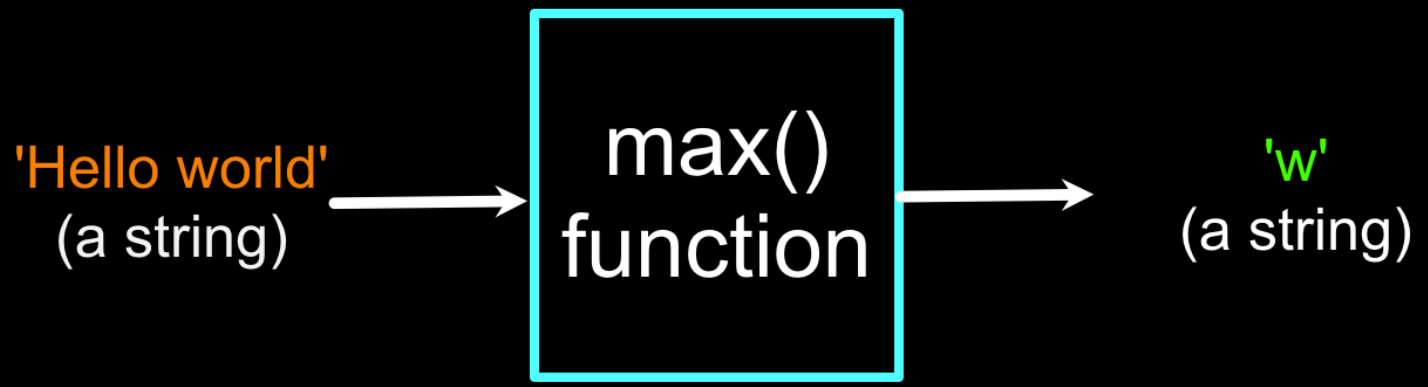



```
>>> big = max('Hello world')
>>> print(big)
w
>>> tiny = min('Hello world')
>>> print(tiny)
```

MAX FUNCTION

- A function is some stored code that we use. A function takes some input and produces an output.

```
>>> big = max('Hello world')
>>> print(big)
w
```



TYPE CONVERSIONS

- When you put an integer and floating point in an expression, the integer is implicitly converted to a float
- You can control this with the built-in functions int() and float()
- Python3: no need for conversion
  print(2*3/4)
  1.5

```
>>> print(float(99) / 100)
0.99
>>> print(99 / 100)
0.99
>>> i = 42
>>> type(i)
<class 'int'>
>>> f = float(i)
>>> print(f)
42.0
>>> f = 1e2
>>> print(f)
100.0
```

STRING CONVERSIONS

- You can also use int() and float() to convert between strings and integers
- You will get an error if the string does not contain numeric characters

```
>>> sval = '123'
>>> type(sval)
<class 'str'>
>>> print(sval + 1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str'
and 'int'
>>> ival = int(sval)
>>> type(ival)
<class 'int'>
>>> print(ival + 1)
124
>>> nsv = 'hello bob'
>>> niv = int(nsv)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int()
```

STRING SUMMATION

- >>> sval1 = '123'
  >>> type(sval)
  <class 'str'>

  >>> sval2 = '456'
  >>> type(sval)
  <class 'str'>

  >>> ssum = sval1 + sval2
  >>> type(ssum)
  <class 'str'>

  >>> print(ssum)
  '123456'

RETURN VALUES

FUNCTION OF YOUR OWN

- Often a function will take its arguments, do some computation, and return a value to be used as the value of the function call in the calling expression. The return keyword is used for this.

```
def greet():
    return "Hello"

print(greet(), "Bob")
print(greet(), "Sarah")
```

```
Hello Bob
Hello Sarah
```

RETURN VALUE

FUNCTION OF YOUR OWN

- A "fruitful" function is one that produces a result (or return value)
- The return statement ends the function execution and "sends back" the result of the function

```
>>> def greet(lang):
...     if lang == 'es':
...         return 'Hola'
...     elif lang == 'fr':
...         return 'Bonjour'
...     else:
...         return 'Hello'
...
>>> print(greet('en'),'Glenn')
Hello Glenn
>>> print(greet('es'),'Sally')
Hola Sally
>>> print(greet('fr'),'Michael')
Bonjour Michael
>>>
```

MULTIPLE PARAMETERS / ARGUMENTS

FUNCTION OF YOUR OWN

- We can define more than one parameter in the function definition
- We simply add more arguments when we call the function
- We match the number and order of arguments and parameters

```
def addtwo(a, b):
    added = a + b
    return added

x = addtwo(3, 5)
print(x)

8
```

LOOPS AND ITERATION