Society of Physics Students

Python Bootcamp

Stuff to talk about

- Data types
- Operators
- Functions
- Terminology
- Flow control
- Packages

Data Types

The main ones

int	5
float	5.0
string	‘five’ “five”
list	[1, 2, 3, 4, ‘5’]
tuple	(x, y)

Other built-ins

dict	{‘key’: value, ‘a’: 5}
bool	True

Data Types

Custom types

numpy's ndarray

astropy's HDUList

Operators

Add	+	“hello” + “there”
Subtract	-	5 - 5
Multiply	*	‘a’ * 5
Divide	/	5 / 5
Modulus	%	3 % 2
Exponent	**	2.718 ** 5
Assignment	=	variable = 5

Operators

Equal	==	`5 == 5`
And	and	`(x > 2) and (x < 3)`
Or	or	`5 or 0`
Not	not	`not True`
Not Equal	!=	`5 != 5`
Greater Than	>	`3 > 2`
Less Than	<	`2 < 5`

Indexing

```
            0    1   2      3       4
my_list = [‘a’,  2,  3,  [1, 1],  6.28]

my_list[0]
>>> ‘a’

my_list[3][0]
>>> 1
```

Functions

`print()`

`len()`

`range()`

`input()`

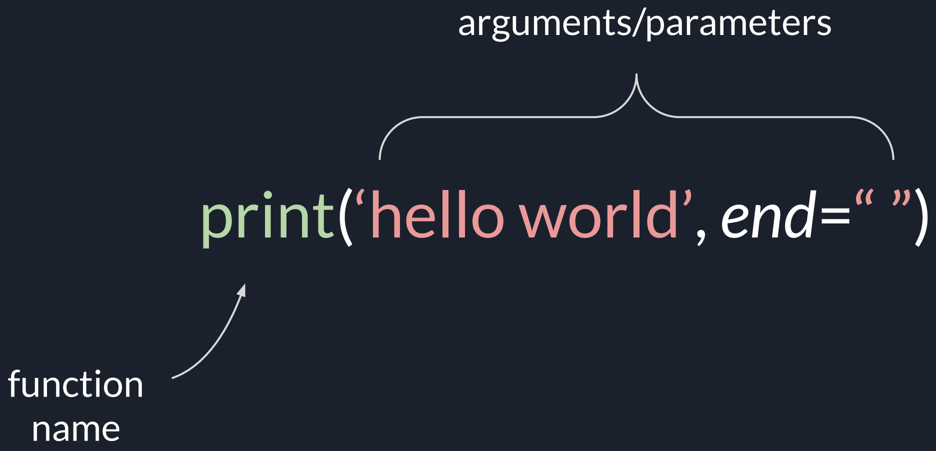
`enumerate()`

`zip()`

`type()`

`list(), int(), str(), float(), etc.`

Terminology

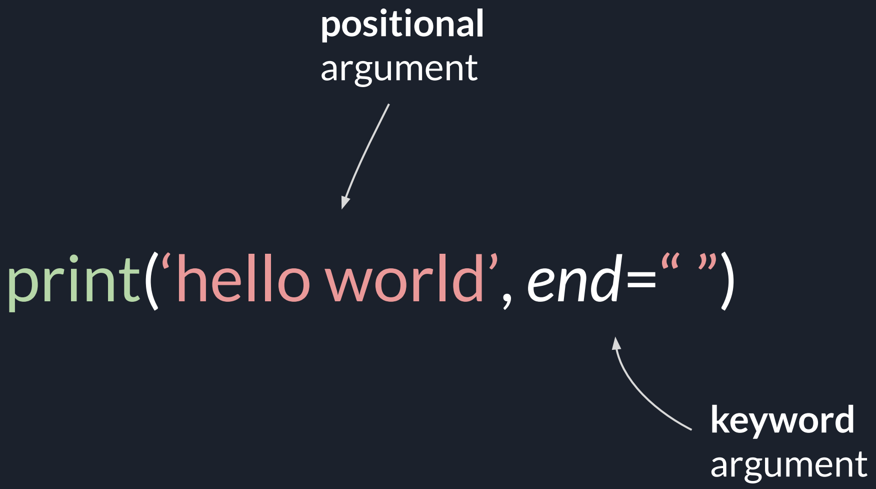


arguments/parameters

```
print(‘hello world’, end=“ ”)
```

function name

Terminology



positional argument

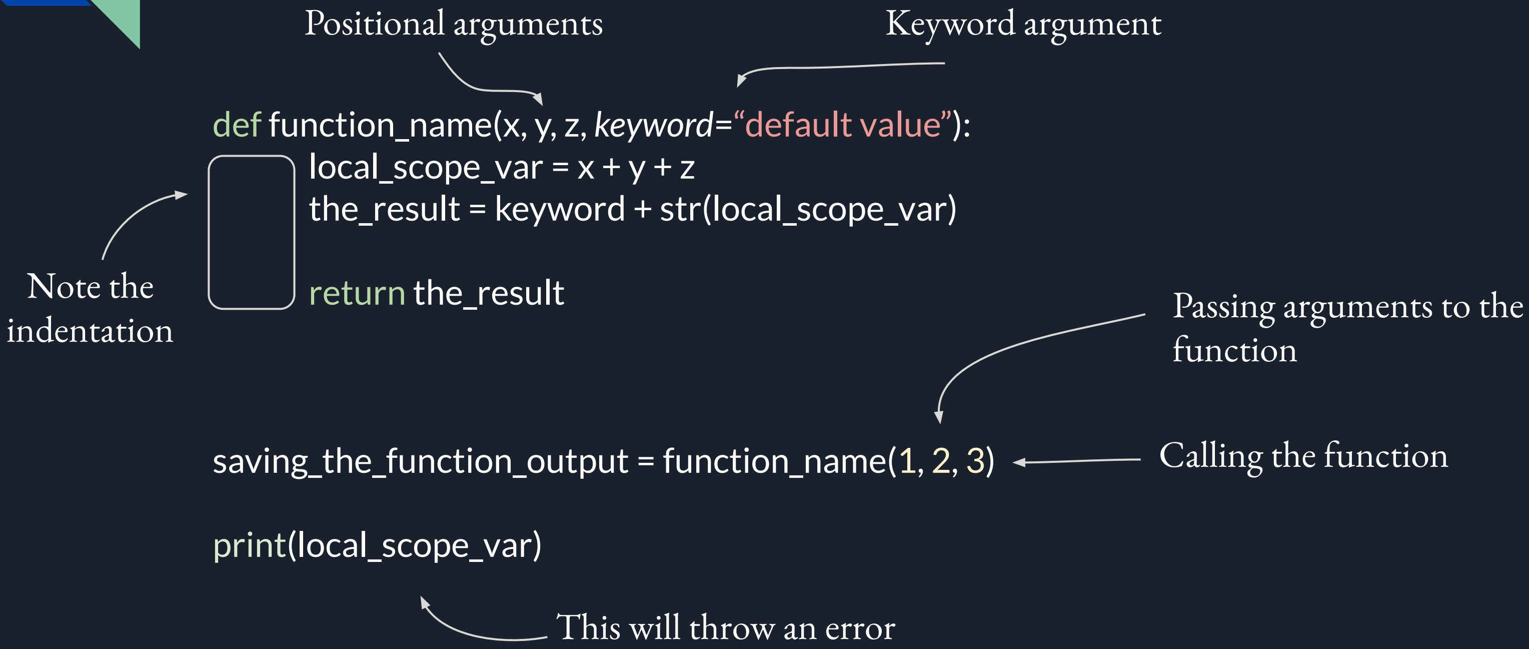
print(‘hello world’, *end*=“ ”)

keyword argument

Defining your own function

```
def function_name(x, y, z, keyword=“default value”):
    local_scope_var = x + y + z
    the_result = keyword + str(local_scope_var)

    return the_result



saving_the_function_output = function_name(1, 2, 3)

print(local_scope_var)
```

Defining your own function



Positional arguments

Keyword argument

```
def function_name(x, y, z, keyword=“default value”):
    local_scope_var = x + y + z
    the_result = keyword + str(local_scope_var)

    return the_result
```

Note the indentation

Passing arguments to the function

```
saving_the_function_output = function_name(1, 2, 3)
```

Calling the function

```
print(local_scope_var)
```

This will throw an error

Flow Control

Conditionals

if statement

- runs code if condition is **True**

elif

- if statement that only gets checked if the preceding if statement fails

else

- runs when all preceding if/elif statements have **False** conditions

Loops

for loop

- iterate over some list
  (or list-like thing)

while loop

- keep running code *while* some condition is **True**

Flow Control

```
M_R_DUCKS = True
C_M_WANGS = False

while M_R_DUCKS:
  if 1:
    print(“This code will always run. Why would anyone write an if statement like this?”)
    for i in range(0, 5, 1):
      print(i ** 2)

  elif (1 + 1) == 2:
    print(“This code will never run”)

  else:
    print(6.283185071795867252)

  if not C_M_WANGS:
    M_R_DUCKS = False
```

Flow Control

```
M_R_DUCKS = True
C_M_WANGS = False

while M_R_DUCKS:
    if 1:
        print(“This code will always run. Why would anyone write an if statement like this?”)
        for i in range(0, 5, 1):
            print(i ** 2)

    elif (1 + 1) == 2:
        print(“This code will never run”)

    else:
        print(6.283185071795867252)

    if not C_M_WANGS:
        M_R_DUCKS = False
```

Colons after flow control statements!

Notice indents

Essentially creates a list of [0, 1, 2, 3, 4]

Packages

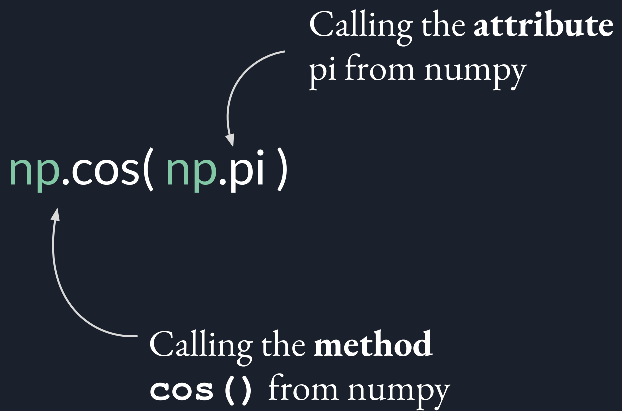
Really commonly used:

- Numpy
- Matplotlib
- Astropy
- Pandas

Importing packages:

```
import datetime

import numpy as np

from astropy.constants import M_sun
```

Packages



Calling the **attribute** pi from numpy

```
np.cos( np.pi )
```

Calling the **method** **`cos()`** from numpy

Use the **.** to access a **method** or an **attribute** from a package

Now it’s your turn!

Open up Colab and let’s get started!