



Python Introduction

Session 1

Python Workshop for Benjama EEP

<http://goo.gl/Zmbzpj>



Objectives

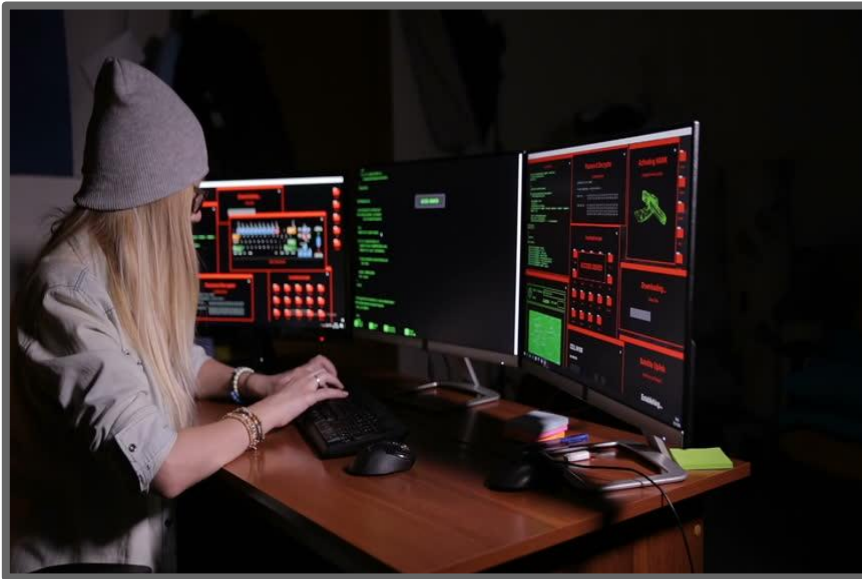
- what is programming?
- variables and data types
- input/output
- module and calling functions

What is programming?





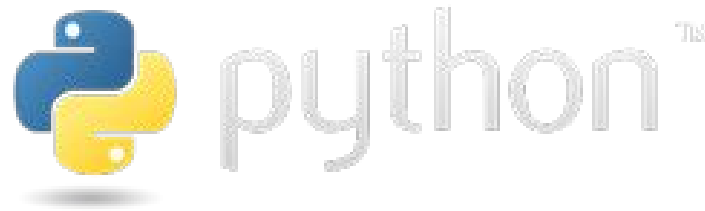
Today, we'll be...



Coding / Implementing



© www.spreadshirt.com



- General purpose
- Object-oriented
- Interpreted
- Focus on readability & productivity
- Strongly typed and dynamically typed

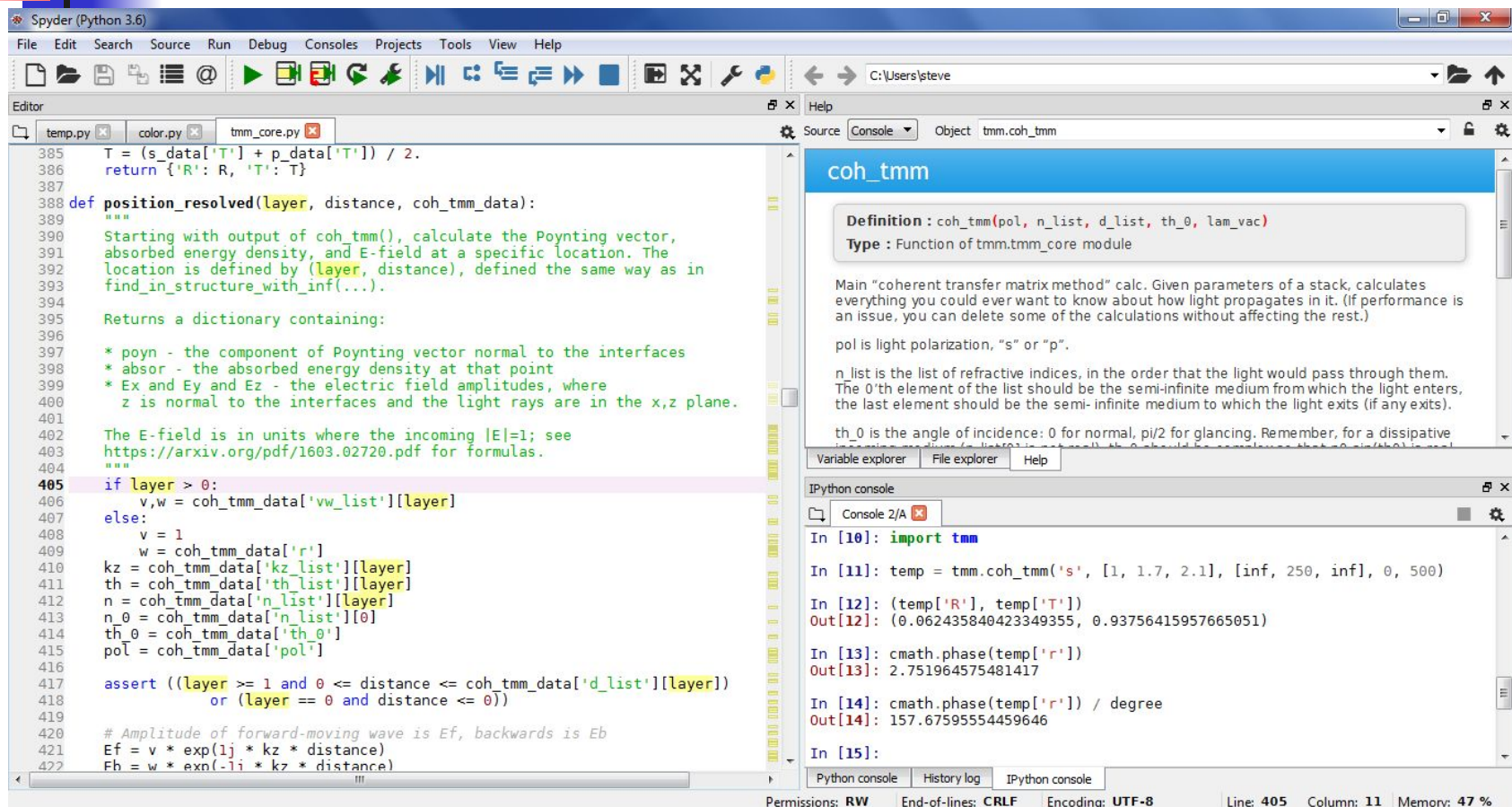


Workshop tools

<https://www.anaconda.com/download>



let's start Spyder - IDE



```
385 T = (s_data['T'] + p_data['T']) / 2.  
386 return {'R': R, 'T': T}  
387  
388 def position_resolved(layer, distance, coh_tmm_data):  
389     """  
390     Starting with output of coh_tmm(), calculate the Poynting vector,  
391     absorbed energy density, and E-field at a specific location. The  
392     location is defined by (layer, distance), defined the same way as in  
393     find_in_structure_with_inf(...).  
394  
395     Returns a dictionary containing:  
396  
397     * poyn - the component of Poynting vector normal to the interfaces  
398     * absor - the absorbed energy density at that point  
399     * Ex and Ey and Ez - the electric field amplitudes, where  
400       z is normal to the interfaces and the light rays are in the x,z plane.  
401  
402     The E-field is in units where the incoming |E|=1; see  
403     https://arxiv.org/pdf/1603.02720.pdf for formulas.  
404     """  
405     if layer > 0:  
406         v,w = coh_tmm_data['vw_list'][layer]  
407     else:  
408         v = 1  
409         w = coh_tmm_data['r']  
410     kz = coh_tmm_data['kz_list'][layer]  
411     th = coh_tmm_data['th_list'][layer]  
412     n = coh_tmm_data['n_list'][layer]  
413     n_0 = coh_tmm_data['n_list'][0]  
414     th_0 = coh_tmm_data['th_0']  
415     pol = coh_tmm_data['pol']  
416  
417     assert ((layer >= 1 and 0 <= distance <= coh_tmm_data['d_list'][layer])  
418           or (layer == 0 and distance <= 0))  
419  
420     # Amplitude of forward-moving wave is Ef, backwards is Eb  
421     Ef = v * exp(1j * kz * distance)  
422     Fb = w * exp(-1j * kz * distance)
```

coh_tmm

Definition: coh_tmm(pol, n_list, d_list, th_0, lam_vac)
Type: Function of tmm.tmm_core module

Main "coherent transfer matrix method" calc. Given parameters of a stack, calculates everything you could ever want to know about how light propagates in it. (If performance is an issue, you can delete some of the calculations without affecting the rest.)

pol is light polarization, "s" or "p".

n_list is the list of refractive indices, in the order that the light would pass through them. The 0th element of the list should be the semi-infinite medium from which the light enters, the last element should be the semi-infinite medium to which the light exits (if any exits).

th_0 is the angle of incidence: 0 for normal, pi/2 for glancing. Remember, for a dissipative medium (e.g. lossy metal) th_0 should be complex, that is, $\theta_0 = \theta_0 + i\alpha$.

Variable explorer | File explorer | Help

IPython console

```
In [10]: import tmm  
  
In [11]: temp = tmm.coh_tmm('s', [1, 1.7, 2.1], [inf, 250, inf], 0, 500)  
  
In [12]: (temp['R'], temp['T'])  
Out[12]: (0.062435840423349355, 0.93756415957665051)  
  
In [13]: cmath.phase(temp['r'])  
Out[13]: 2.751964575481417  
  
In [14]: cmath.phase(temp['r']) / degree  
Out[14]: 157.67595554459646  
  
In [15]:
```

Python console | History log | IPython console

Permissions: RW | End-of-lines: CRLF | Encoding: UTF-8 | Line: 405 | Column: 11 | Memory: 47 %

Spyder - IDE

The image shows the Spyder Python IDE interface. The top panel is the **Editor**, which contains a Python script named `tmm_core.py`. The script defines a function `position_resolved` that calculates the Poynting vector, absorbed energy density, and E-field at a specific location. The function takes `layer`, `distance`, and `coh_tmm_data` as arguments. The script also includes a `coh_tmm` module definition and a `main` function. The `position_resolved` function is highlighted with a yellow background. The `coh_tmm` module definition is highlighted with a blue background. The `main` function is highlighted with a pink background. The `position_resolved` function is highlighted with a yellow background. The `coh_tmm` module definition is highlighted with a blue background. The `main` function is highlighted with a pink background.

The middle panel is the **Help** panel, which displays the documentation for the `coh_tmm` module. The documentation includes the definition of the `coh_tmm` function, its type, and a detailed description of its parameters and return values. The `coh_tmm` module definition is highlighted with a blue background.

The bottom panel is the **Console** panel, which displays the output of the Python script. The console shows the execution of the `main` function, which calls the `position_resolved` function. The console output is highlighted with a yellow background.

The **Editor** panel shows the following code:

```
385 T = (s_data['T'] + p_data['T']) / 2.  
386 return {'R': R, 'T': T}  
387  
388 def position_resolved(layer, distance, coh_tmm_data):  
389     """  
390     Starting with output of coh_tmm(), calculate the Poynting vector,  
391     absorbed energy density, and E-field at a specific location. The  
392     location is defined by (layer, distance), defined the same way as in  
393     find_in_structure_with_inf(...).  
394  
395     Returns a dictionary containing:  
396  
397     * poyn - the component of Poynting vector normal to the interfaces  
398     * absor - the absorbed energy density at that point  
399     * Ex and Ey and Ez - the electric field amplitudes, where  
400       z is normal to the surface and the light rays are in the x,z plane.  
401  
402     The E-field is in units where  $E_0 = 1$  V/m. See  
403     https://arxiv.org/pdf/1603.02001v1.pdf for details.  
404  
405     if layer > 0:  
406         v,w = coh_tmm_data['vw_list'][layer]  
407     else:  
408         v = 1  
409         w = coh_tmm_data['r']  
410     kz = coh_tmm_data['kz_list'][layer]  
411     th = coh_tmm_data['th_list'][layer]  
412     n = coh_tmm_data['n_list'][layer]  
413     n_0 = coh_tmm_data['n_list'][0]  
414     th_0 = coh_tmm_data['th_0']  
415     pol = coh_tmm_data['pol']  
416  
417     assert ((layer >= 1 and 0 <= distance <= coh_tmm_data['d_list'][layer])  
418           or (layer == 0 and distance <= 0))  
419  
420     # Amplitude of forward-moving wave is Ef, backwards is Eb  
421     Ef = v * exp(1j * kz * distance)  
422     Fb = w * exp(-1j * kz * distance)
```

The **Help** panel shows the following documentation for the `coh_tmm` module:

Definition: `coh_tmm(pol, n_list, d_list, th_0, n_vac)`
Type: Function of `tmm.tmm_core` module

Main "coherent transfer matrix" method. Calculates everything you could ever want to know about how light propagates in it. (If performance is an issue, you can delete some of the calculations with `delete` affecting the rest.)

`pol` is light polarization, "s" or "p".

`n_list` is the list of refractive indices, in the order that the light would pass through them. The 0'th element of the list should be the semi-infinite medium from which the light enters, the last element should be the semi-infinite medium to which the light exits (if any exits).

The **Console** panel shows the following output:

```
In [10]: import tmm  
In [11]: temp = tmm.coh_tmm('s', [1, 1.7, 2.1], [inf, 0, 500])  
In [12]: (temp['R'], temp['T'])  
Out[12]: (0.062431404233401, 0.937568595766598)  
In [13]: cmath.phase(temp['T'])  
Out[13]: 2.751964575481417  
In [14]: cmath.phase(temp['R']) / degree  
Out[14]: 157.67595554459646  
In [15]:
```



Code - app101.py

- เขียนและทดสอบโปรแกรมแรกได้
 - Edit & Run

```
print('สวัสดี')
```



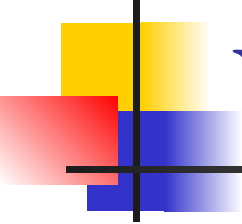
Code - app102.py

■ เรียกใช้คำสั่งเพื่อให้ผู้ใช้กรอกข้อมูล

- `input()`
- `input('กรอกชื่อ ')`
- `input('กรอกส่วนสูง ')`

print ('สวัสดี')

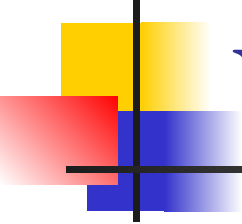
`input('กรอกชื่อ ')`



Variables - app103.py

- กำหนดชื่อเพื่ออ้างถึงข้อมูลที่ใช้กรอก
- นำมาแสดงผล

```
name = input('กรอกชื่อ ')\nprint('สวัสดี ')\nprint(name)
```



Variables - app104.py

- การแสดงข้อมูลติดกัน

```
name = input('กรอกชื่อ ')\nprint('สวัสดี คุณ'+name)
```



Variables - ชนิดของข้อมูล

■ **ข้อความ** - ตัวอักษรตัวเลขเรียงติดกันอยู่ระหว่าง ‘ ’

■ str

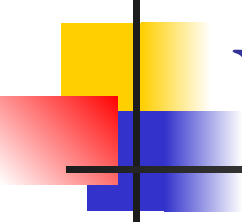
■ ‘กรอกชื่อ ’

■ “สวัสดีครับ Dr.Tom & Jerry”

■ **ตัวเลข** - ตัวเลขที่สามารถนำมาคำนวณได้

■ 20 → int

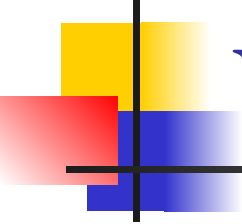
■ 3.55 → float



Variables - app105.py

- รับชื่อและรับตัวเลขจำนวนเต็ม

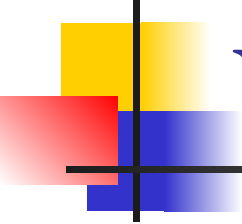
```
name = input('กรอกชื่อ ')\nprint('สวัสดี คุณ'+name)\nyear = int(input('เกิดเมื่อ พ.ศ. '))\nprint('ปีเกิด '+str(year))
```



Variables - app106.py

- รับชื่อและรับตัวเลขทศนิยม

```
name = input('กรอกชื่อ ')\nprint('สวัสดี คุณ'+name)\ngpa = float(input('เกรดเฉลี่ย '))\nprint('เกรดเฉลี่ย '+str(gpa))
```

Variables - app107.py

- รับชื่อและปีพ.ศ.เกิด คำนวณอายุ

```
name = input('กรอกชื่อ ')\nprint('สวัสดี คุณ'+name)\nyear = int(input('ปีเกิด พ.ศ. '))\nage = 2561-year\nprint('คุณอายุ '+str(age))
```



Operators

■ คูณ - ยกกำลัง

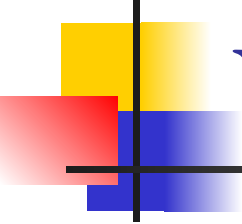
■ $4 * 5$ $2 ** 8$

■หาร - หารตัดเศษ - หารเอาเศษ

■ $9 / 4$ $9 // 4$ $9 \% 4$

■บวก - ลบ

■ $9 + 4$ $9 - 4$



Variables - app108.py

■ คำนวณหาความชัน

```
x1 = float(input('x1 '))  
y1 = float(input('y1 '))  
x2 = float(input('x2 '))  
y2 = float(input('y2 '))  
m = (y2-y1) / (x2-x1)  
print('ความชัน ' + str(m))
```



Exercise

1. โปรแกรมแปลงขนาดที่ดินจาก*ตารางวา*เป็น*ตารางเมตร*
2. โปรแกรมหาพื้นฐานของวงกลมโดยให้ผู้ใช้กรอกรัศมี
(ใช้ $\pi = 22 / 7$)
3. โปรแกรมแปลงอุณหภูมิจาก *Fahrenheit* เป็น *Celsius*
(สมการ $F = C * 9/5 + 32$)



Variables - list

■ ลำดับของ **str**

```
speakers = [ 'K', 'Kai', 'Or', 'Paul', 'Tom', 'Woot' ]
```

■ ลำดับของ **int**

```
heights = [ 180, 164, 165, 165, 180, 170 ]
```

■ ลำดับของ **float**

```
popularity = [ 3.25, 3.9, 4, 3, 3.5, 3.44 ]
```



Speakers in one line - app109.py

- รับชื่อวิทยากรที่อยู่ใน 1 บรรทัด

```
line = input('กรอกรายชื่อวิทยากร ')\nprint('line '+line)\nspeakers = line.split(' ')\nprint('speakers '+str(speakers))\nหรือ\nprint(str(speakers))
```



Speakers Data - app110.py

- อ่านข้อมูลวิทยากรที่เป็นลำดับของตัวเลข

```
line = input('ส่วนสูง ')\nheight = map(int, line.split(' '))\nprint(height)
```

```
line = input('pop ')\npop = map(float, line.split(' '))\nprint(pop)
```



Module and Calling Functions

- จะคำนวณหา ค่าเฉลี่ย ค่ากลางอย่างไร?
 - เรียกใช้ module เสริมชื่อ statistics
 - `import statistics`
- หลังจากที่ได้ข้อมูลจากผู้ใช้แล้วเรียกใช้
 - `statistics.median(height)`
 - `statistics.mean(pop)`
 - `statistics.mode() / median() / stdev()`



Statistics - app111.py

```
import statistics
```

```
line = input('speakers ')  
speakers = map(str, line.split(' '))  
print(speakers)
```

```
line = input('height ')  
height = map(int, line.split(' '))  
print(statistics.mode(height))
```

```
line = input('pop ')  
pop = map(float, line.split(' '))  
print(statistics.mean(pop))
```