

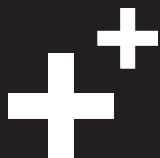
# Python开发进阶

**NSD PYTHON2**

**DAY01**

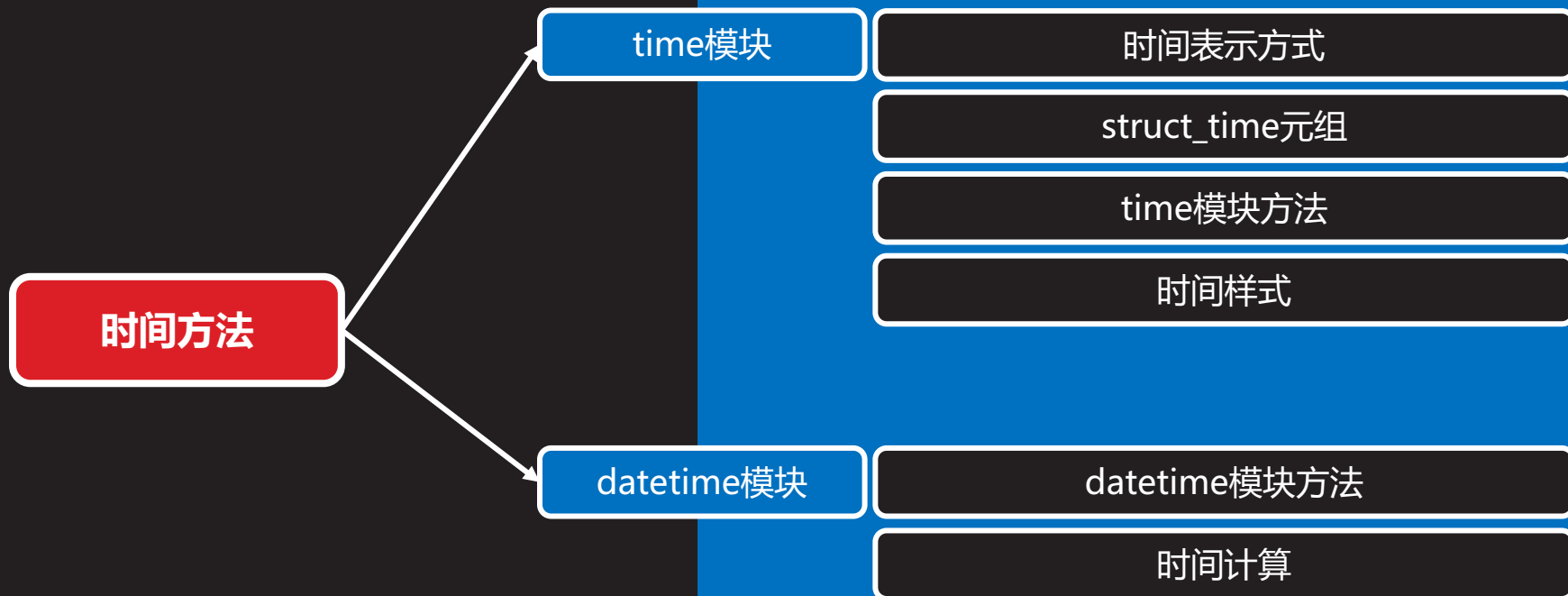
# 内容

上午	09:00 ~ 09:30	时间方法
	09:30 ~ 10:20	
	10:30 ~ 11:20	异常处理
	11:30 ~ 12:00	
下午	14:00 ~ 14:50	文件系统相关模块
	15:00 ~ 15:50	
	16:10 ~ 17:00	
	17:10 ~ 18:00	总结和答疑



# 时间方法

---



# time模块

---

# 时间表示方式

- 时间戳timestamp：表示的是从1970年1月1日00:00:00开始按秒计算的偏移量
- UTC ( Coordinated Universal Time , 世界协调时 ) 亦即格林威治天文时间，世界标准时间。在中国为UTC+8。DST ( Daylight Saving Time ) 即夏令时
- 元组 ( struct\_time ) ：由9个元素组成



# struct\_time元组

索引	属性	值
0	tm_year	2000
1	tm_mon	1-12
2	tm_mday	1-31
3	tm_hour	0-23
4	tm_min	0-59
5	tm_sec	0-61
6	tm_wday	0-6(0表示周一)
7	tm_yday(一年中的第几天)	1-366
8	tm_isdst(是否为dst时间)	默认为-1



# time模块方法

- `time.localtime([secs])` : 将一个时间戳转换为当前时区的`struct_time`。secs参数未提供, 则以当前时间为准
- `time.gmtime([secs])` : 和`localtime()`方法类似, `gmtime()`方法是 将一个时间戳转换为UTC时区 ( 0 时区 ) 的`struct_time`
- `time.time()` : 返回当前时间的时间戳
- `time.mktime(t)` : 将一个`struct_time`转化为时间戳



## time模块方法（续1）

- `time.sleep(secs)`：线程推迟指定的时间运行。单位为秒
- `time.asctime([t])`：把一个表示时间的元组或者 `struct_time` 表示为这种形式：'Sun Jun 20 23:21:05 1993'。如果没有参数，将会将 `time.localtime()` 作为参数传入
- `time.ctime([secs])`：把一个时间戳（按秒计算的浮点数）转化为 `time.asctime()` 的形式





## time模块方法（续2）

- `time.strftime(format[, t])`：把一个代表时间的元组或者`struct_time`（如由`time.localtime()`和`time.gmtime()`返回）转化为格式化的时间字符串。如果`t`未指定，将传入`time.localtime()`
- `time.strptime(string[, format])`：把一个格式化时间字符串转化为`struct_time`。实际上它和`strftime()`是逆操作

```
>>> time.strftime('%Y-%m-%d %X', time.localtime())
'2017-12-12 12:58:19'
```



# 时间样式

格式	含义	格式	含义
%a	本地简化星期名称	%m	月份（01 - 12）
%A	本地完整星期名称	%M	分钟数（00 - 59）
%b	本地简化月份名称	%p	本地am或者pm的相应符
%B	本地完整月份名称	%S	秒（01 - 61）
%c	本地相应的日期和时间	%U	一年中的星期数（00 - 53，星期日是一个星期的开始）
%d	一个月中的第几天（01 - 31）	%w	一个星期中的第几天（0 - 6，0是星期天）
%H	一天中的第几个小时（24小时制，00 - 23）	%x	本地相应日期
%I	第几个小时（12小时制，01 - 12）	%X	本地相应时间
%j	一年中的第几天（001 - 366）	%y	去掉世纪的年份（00 - 99）
%Z	时区的名字	%Y	完整的年份



# datetime模块

---

# datetime模块方法

- `datetime.today()` : 返回一个表示当前本地时间的 `datetime` 对象
- `datetime.now([tz])` : 返回一个表示当前本地时间的 `datetime` 对象, 如果提供了参数 `tz`, 则获取 `tz` 参数所指时区的本地时间
- `datetime.strptime(date_string, format)` : 将格式字符串转换为 `datetime` 对象
- `datetime.ctime(datetime对象)` : 返回时间格式字符串
- `datetime.strftime(format)` : 返回指定格式字符串



# 时间计算

- 使用timedelta可以很方便的在日期上做天days，小时hour，分钟，秒，毫秒，微妙的时间计算

```
>>> dt = datetime.datetime.now()
>>> days = datetime.timedelta(days=100, hours=3)
>>> dt + days
datetime.datetime(2050, 6, 10, 20, 41, 20, 106546)
```



# 异常处理

## 异常处理

### 异常

什么是异常

python中的异常

try-except语句

带有多个except的try语句

捕获所有异常

异常参数

else子句

finally子句

### 异常处理

raise语句

断言

# 异常



# 什么是异常

- 当python检测到一个错误时，解释器就会指出当前流已经无法继续执行下去，这时候就出现了异常
- 异常是因为程序出现了错误而在正常控制流以外采取的行为
- 这个行为又分为两个阶段：
  - 首先是引起异常发生的错误
  - 然后是检测（和采取可能的措施）阶段





# python中的异常

- 当程序运行时，因为遇到未解的错误而导致中止运行，便会出现traceback消息，打印异常

异常	描 述
NameError	未声明/初始化对象
IndexError	序列中没有没有此索引
SyntaxError	语法错误
KeyboardInterrupt	用户中断执行
EOFError	没有内建输入，到达EOF标记
IOError	输入/输出操作失败



# try-except语句

- 定义了进行异常监控的一段代码，并且提供了处理异常的机制

```
try:
    try_suite #监控这里的异常
except Exception[as reason]:
    except_suite #异常处理代码
```

```
>>> try:
...     f = open('foo.txt')
... except FileNotFoundError:
...     print('No such file')
...
No such file
```



# 带有多个except的try语句

- 可以把多个except语句连接在一起，处理一个try块中可能发生的多种异常

```
>>> try:
...     data = int(input('input a number: '))
... except KeyboardInterrupt:
...     print 'user cancelled'
... except ValueError:
...     print('you must input a number!')
...
input a number: hello
you must input a number!
```



# 异常参数

- 异常也可以有参数，异常引发后它会被传递给异常处理器
- 当异常被引发后参数是作为附加帮助信息传递给异常处理器的

```
>>> try:
...     10 / 0
... except ZeroDivisionError as e:
...     print('error', e)
...
error division by zero
```



# 案例1：简化除法判断

1. 提示用户输入一个数字作为除数
2. 如果用户按下Ctrl+C或Ctrl+D则退出程序
3. 如果用户输入非数字字符，提示用户应该输入数字
4. 如果用户输入0，提示用户0不能作为除数



# else子句

- 在try范围中没有异常被检测到时，执行else子句
- 在else范围中的任何代码运行前，try范围中的所有代码必须完全成功

```
>>> try:
...     result = 100 / int(input("number: "))
... except Exception as e:
...     print('Error:', e)
... else:
...     print(result)
...
number: 10
10.0
```



# finally子句

- finally子句是无论异常是否发生，是否捕捉都会执行的一段代码
- 如果打开文件后，因为发生异常导致文件没有关闭，可能会发生数据损坏。使用finally可以保证文件总是能正常的关闭



# 触发异常





# raise语句

- 要想引发异常，最简单的形式就是输入关键字raise，后面跟要引发的异常的名称
- 执行raise语句时，Python会创建指定的异常类的一个对象
- raise语句还可指定对异常对象进行初始化的参数



# 断言

- 断言是一句必须等价于布尔值为真的判定
- 此外，发生异常也意味着表达式为假

```
>>> assert 10 > 100, "Wrong"  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
AssertionError: Wrong
```



## 案例2：自定义异常

1. 编写第一个函数，函数接收姓名和年龄，如果年龄不在1到120之间，产生ValueError异常
2. 编写第二个函数，函数接收姓名和年龄，如果年龄不在1到120之间，产生断言异常



# OS相关模块

---



# os模块



# os模块简介

- 对文件系统的访问大多通过python的os模块实现
- 该模块是python访问操作系统功能的主要接口
- 有些方法，如copy等，并没有提供，可以使用shutil模块作为补充



# os模块方法

函数	作 用
symlink()	创建符号链接
listdir()	列出指定目录的文件
getcwd()	返回当前工作目录
mkdir()	创建目录
chmod()	改变权限模式
getatime()	返回最近访问时间
chdir()	改变工作目录



## 案例3：操作文件系统

- 编写脚本，熟悉os模块操作
  1. 切换到/tmp目录
  2. 创建example目录
  3. 切换到/tmp/example目录
  4. 创建test文件，并写入字符串foo bar
  5. 列出/tmp/exaple目录内容
  6. 打印test文件内容
  7. 反向操作，把test文件以及example目录删除





# pickle模块

---

# pickle模块简介

- 把数据写入文件时，常规的文件方法只能把字符串对象写入。其他数据需先转换成字符串再写入文件。
- python提供了一个标准的模块，称为pickle。使用它可以在一个文件中储存任何python对象，之后又可以把它完整无缺地取出来



# pickle模块方法

- 分别调用dump()和load()可以存储、写入

```
>>> import pickle as p
>>> shoplistfile = 'shoplist.data'
>>> shoplist = ['apple', 'mango', 'carrot']
>>> f = file(shoplistfile, 'wb')
>>> p.dump(shoplist, f)
>>> f.close()
>>>
>>> f = file(shoplistfile)
>>> storedlist = p.load(f)
>>> print storedlist
['apple', 'mango', 'carrot']
```



## 案例4：记账程序

1. 假设在记账时，有一万元钱
2. 无论是开销还是收入都要进行记账
3. 记账内容包括时间、金额和说明等
4. 记账数据要求永久存储



# 总结和答疑

---